commodore 16

USER MANUAL

commodore

1 Hunters Road, Weldon, Corby, Northants NN17 1QX

commodore

COMMODORE 16
USER MANUAL

TABLE OF CONTENTS

CHAPTER 1

SETTING UP

- Unpacking your Commodore 16
- Getting to know the switches and sockets
- Setting up your Commodore 16
- Troubleshooting chart
- Peripherals

UNPACKING YOUR COMMODORE 16

Now that you've opened the box containing your new Commodore 16 and found this manual, the first thing that you should do is check to make sure that you have all the items on this list. You should have:

1. Your Commodore 16
2. The power supply
3. The TV cable
4. The user manual (You've probably found this, since you're reading it right now.)
5. Other assorted literature:
   Warranty card

If you don't find all these items in the box, check with your dealer immediately for replacements.

Before you connect anything, you should overlook these pictures of your computer. You can familiarize yourself with all the outlets and switches, so you can set up your computer system quickly and easily.

GETTING TO KNOW THE SWITCHES AND SOCKETS

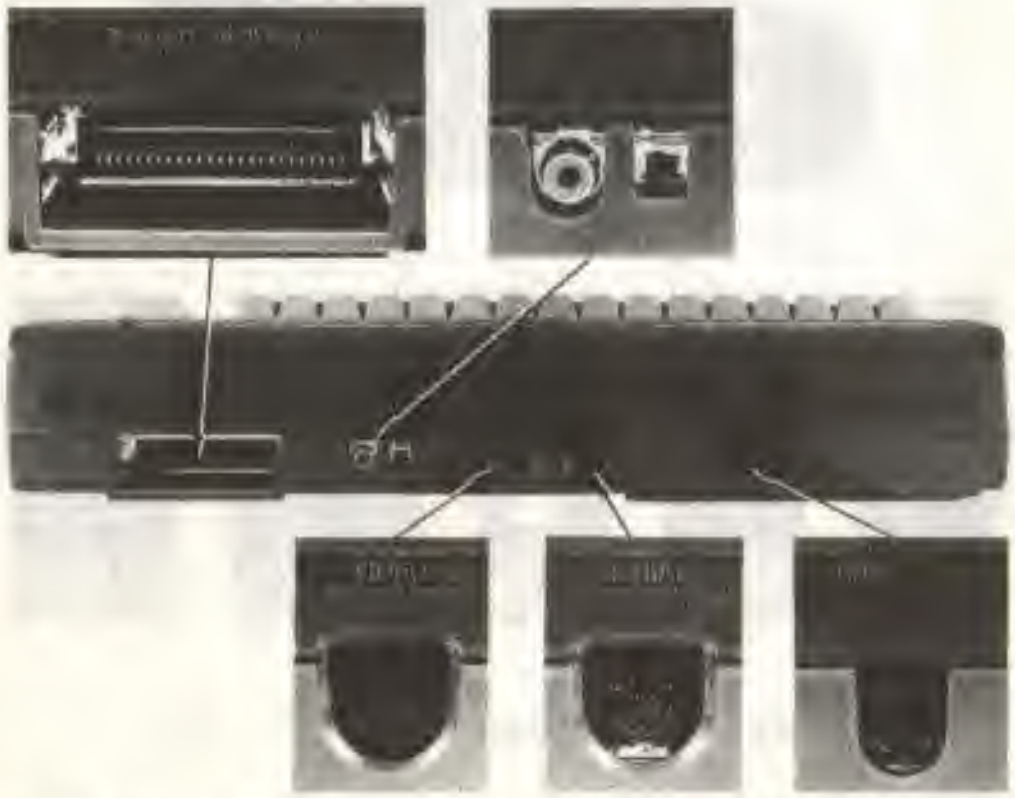
The Right Side Of Your Commodore 16

THE ON/OFF SWITCH
Your Commodore 16 should be turned **OFF** when you install or remove cartridges or any peripheral device such as a printer or Datassette. There is a red power light located above the keyboard on the right, which lights up when power is on.

THE JOYSTICK SOCKETS
You can plug joysticks into these sockets, labelled JOY #1 and JOY #2. The Commodore 16 uses specially designed joysticks available from your Commodore dealer.

THE RESET BUTTON
Press the reset button when you want to "start over," as if you'd just turned your computer on. The reset button clears the screen and erases any BASIC programs that were typed in. There are other ways to reset your computer without erasing programs. These will be discussed in Chapter 4.

THE POWER SOCKET
The end of the power supply cable fits in here. Plug the other end into a standard electrical wall socket **AFTER** connecting the first end into your computer.



The Back Of Your Computer

THE MEMORY EXPANSION PORT
Commodore 16 software cartridges plug in here. Before you insert or remove cartridges, make sure the power is OFF.

THE RF JACK
This is where you plug in one end of the TV cable (the thin black cable). Only one end fits into this jack; the other end fits into the TV serial socket.

THE HIGH/LOW SWITCH
This switch controls which television channel your Commodore 16 is on. This is primarily used for American TV's.

THE VIDEO SOCKET
This is where you plug in the cable that connects a monitor to your Commodore 16. If you hook up your computer to a television set, you won't need to use this.

THE SERIAL SOCKET
You can plug a disk drive or a printer into this socket. If you want to plug in both, first plug the disk drive into this opening, and then plug the printer cable into the back of the disk drive.

THE CASSETTE PORT
The Commodore 1531 Datassette tape recorder for cassette tape software plugs in here.

SETTING UP FOR COMMODORE 16

Connecting your Commodore 16 is simple. You only need to do two things:

1 Plug in the POWER SUPPLY on the right side of your computer and then into a wall plug.

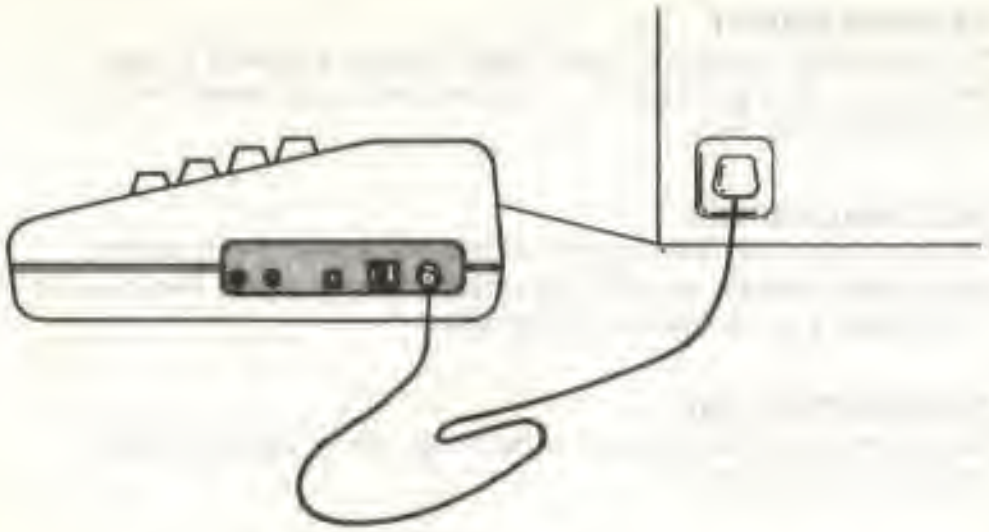
2 Connect the TV cable (the thin black one) from the aerial socket on your TV to the RF SOCKET on the back of your computer.

Make sure that you have enough electrical wall outlets to plug everything in near where you decide to set up your computer. You may need a power strip or extension cord for more outlets, especially if you're also hooking up a printer or a disk drive. Remember that the power on everything (computer, TV, etc.) should be turned OFF until you are completely set up and ready to go.

1: Connecting The Power Supply

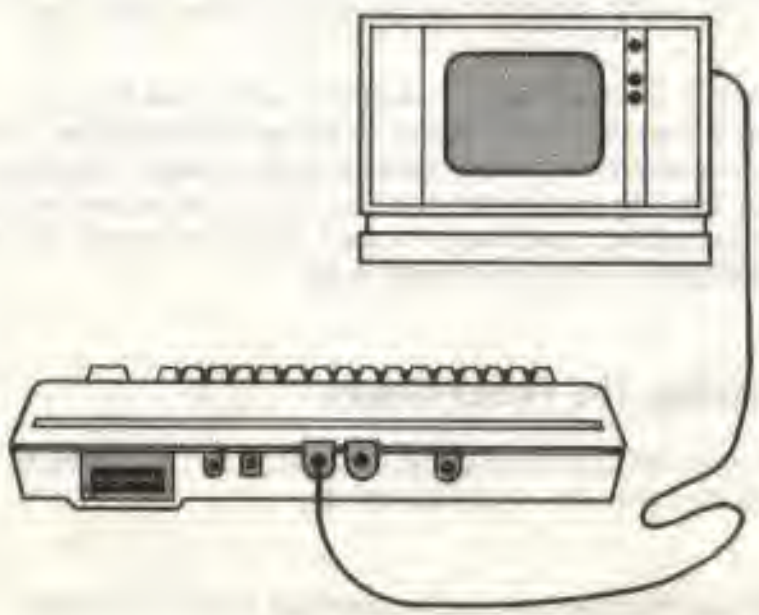
- FIRST plug the round end of the POWER SUPPLY into the power jack on the right side of your computer.

- THEN plug the other end into an electrical socket on the wall.



2: Plugging In The TV Cable

- Plug one end of the TV cable (only one will fit) into the serial socket on your TV.
- Plug the other end of the cable into the RF jack on the back of your computer.



Connecting Your Commodore 16 To A Monitor

If you're connecting your computer to a monitor instead of a TV, follow the instructions in the manual that is included with the monitor. Hooking up a monitor like the Commodore 1702 Colour Monitor is simple. You need to connect only one cable which goes directly from your monitor to the VIDEO socket on the back of your computer.

Finally . . .

Now it's time to turn on your computer. (If you've been paying attention, you should know where the POWER switch is by now.)

If all went well, the red POWER light goes on, and this message appears on your screen

```
COMMODORE BASIC V3.5 12277 BYTES FREE
READY.
```

The flashing cursor under the READY message tells you that your Commodore 16 is waiting for you to start typing. The background colour is white, while the letters are printed in black, with a light purple border around the screen.

If all didn't go well, the Troubleshooting Chart should come in handy.

TROUBLESHOOTING CHART

Symptom	Cause	Remedy
Indicator light not 'ON'	Computer not turned ON	Make sure power switch is in ON position
	Power cable not plugged in	Check power socket for loose or disconnected power cable
	Power supply not plugged in	Check connection with wall outlet
	Bad fuse in computer	Take system to authorised dealer for replacement of fuse

Symptom	Cause	Remedy
No picture	TV cable not plugged in	Check TV cable connection
	TV not on	Turn TV on
Random pattern on TV with cartridge in place	Cartridge not properly inserted	Reinsert cartridge after turning OFF power
Picture without colour	Poorly tuned TV	Retune TV
	TV not connected properly	Check connections
	Colour set too low on TV or computer	Adjust colour setting
Picture OK, but no sound	TV volume too low	Adjust volume of TV
	Poorly tuned TV	Retune TV

IMPORTANT: Some TV sets cannot display the entire Commodore 16 screen. Instead, their picture cuts off the far left and far right column of the screen. We recommend using a different TV set or a monitor such as the Commodore 1702, 1802 or 1803 colour monitor.

If this isn't possible, you can deal with the problem by pressing the ESC key, followed by the 'R' key. This reduces the computer screen display size to 38 columns, so that the entire picture can fit onscreen. You must repeat this each time you power up or reset your computer.

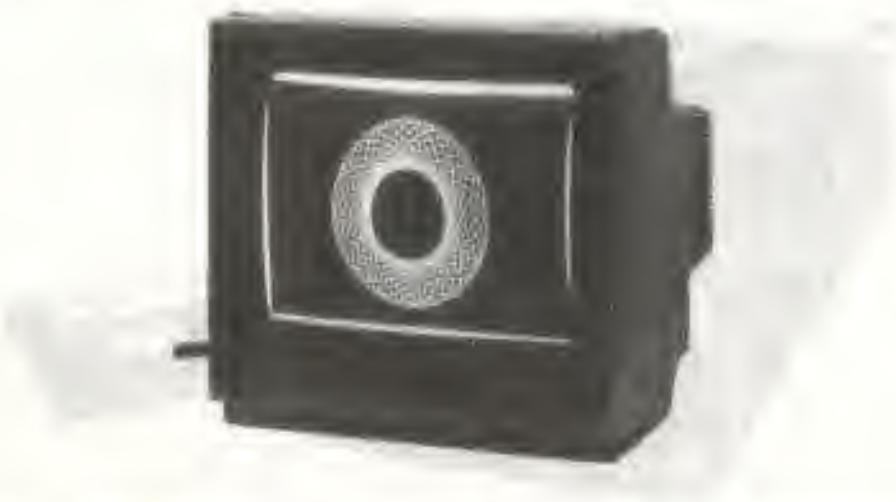
PERIPHERALS

Peripherals are the accessories that you can get to go with your Commodore 16 that increase what you can do with your computer. These accessories are available at your Commodore dealer, and allow you to use your computer to the fullest.

Peripherals give you the capability to save and store data, print out what appears onscreen (in black and white or colour), use software programs that are stored on cassette tape and floppy diskette, and give you a sharp, clear picture of your computer's display.

To save or recall programs, you'll need a device that stores data. Data can be recorded on and retrieved from both cassette tapes and diskettes. To use cassette tape software (and to record your own programs on cassettes), you'll need the Commodore 1531.

DATASSETTE tape recorder. For diskettes, there are several DISK DRIVES that are suitable. Disk drives are typically fast and efficient to use. Disk drives that are compatible with your Commodore 16 are the Commodore models 1541 and 1551.



Your television set may not give you as clear a picture as you'd like for your computer. Commodore colour monitors are specially designed to give you the sharpest, brightest picture for viewing your Commodore 16 output. There are several models available, including the Commodore 1702, 1802 and 1803.

When using a wordprocessing program or a graphics package on your Commodore 16, a printer will reproduce what is on the screen on paper. There are several models of Commodore printers available that work with your computer. These include the MPS-801, MPS-802, MPS-803 (with optional tractor-feed), and DPS-1101 (letter quality).

CHAPTER 2

THE KEYBOARD

- A tour of the keyboard
- Special keys
- Graphic keys
- Function keys
- The HELP key

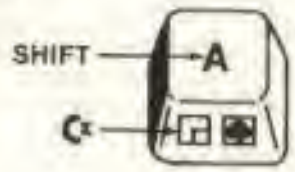
A TOUR OF THE KEYBOARD

Most of the keys on your Commodore 16 keyboard are identical to the keys on a typewriter, but each key can do more than a typewriter key. In this section, you'll learn how to use special keys like the C= key and the four separate cursor keys. This section will show you the extra features of every key, including how to use the graphic symbols pictured on the fronts of many of the keys. With each explanation of the different keys on your computer keyboard, you should find the keys and practice using them.

Using Your Keyboard Like a Typewriter

When you first type letters on your computer, they appear as capitals on the screen. The letters and numbers appear on the screen exactly as they appear on the face of the key when you press the key by itself. Also, several other keys (+, −, =, @, *, and £) may be typed alone. Most punctuation marks need to be typed with the **SHIFT** key. If you want to do some "regular" typing, you can type in capital and lowercase letters (like you would on a typewriter) by pressing the **SHIFT** key and the C= key at the same time. After you do this, all letter keys typed alone are printed in lowercase. When you press the **SHIFT** key along with a letter, you get a capital letter. Numbers and punctuation keys work the same as they would in regular (non-typing) mode. To get out of typing mode, just press the two keys (**SHIFT** and C=) together again.

KEY	+	KEY	=	
SHIFT	+	C=	=	ENTER OR EXIT TYPING MODE
A			=	a
SHIFT	+	A	=	A
C=	+	A	=	┌



Special Keys

Several keys on your Commodore 16 keyboard behave quite differently than any well brought-up typewriter key would even think of acting. These keys act to enable other keys to do things they wouldn't ordinarily do, or perform functions related to the business of computing. Your Commodore 16 keyboard also contains special symbols not found on many typewriters, or even on most computers. These special symbols include the pound sign (£), pi (π), greater and less than signs (< >), brackets ([ ]), and arrows (← ↑). These special symbols keys are often used in writing programs on your Commodore 16.

RETURN

You have to press the **RETURN** key at the end of each line of instructions you enter on your Commodore 16 keyboard. You might think of this key as an ENTER key because **RETURN** actually enters information and instructions into your computer.

SHIFT

You've already come across an example of the **SHIFT** key in action in using your keyboard like a typewriter. That is typical of the **SHIFT** key: it is always used to modify what other keys print onscreen, but can't do anything by itself. (Always the bridesmaid, never the bride . . .) The **SHIFT** key allows you to type capital letters, graphic symbols, punctuation marks, and a few other things—with a little help from another key. You'll be seeing more of the **SHIFT** key's functions throughout this section, for things such as getting graphic symbols.

The **SHIFT LOCK** key is the same as the **SHIFT** key, except that it is locked into place, so you don't have to hold it down. When **SHIFT LOCK** is on, every character you type is SHIFTED. To release the lock, just press **SHIFT LOCK** again, and everything you type is back to normal.

RUN/STOP

Press this key to break into a running program to STOP what your Commodore 16 is doing. When your computer is running a program, pressing this key gets you back in control of the keyboard.

When you hold down the **SHIFT** and **RUN/STOP** keys simultaneously, the Commodore 16 loads and runs the first program on a disk in the disk drive.

The Cursor Keys

The cursor, the flashing block that marks where you are on the computer screen, can be moved quickly and easily around the screen by using the CURSOR KEYS. There are four separate cursor keys, each with an arrow pointing out the direction the key moves the cursor: up, down, left, or right. You can use the cursor keys to move the cursor over anything on the screen without affecting those characters. Like all keys on the Commodore 16 keyboard, each cursor key can automatically repeat. This means that if you continually hold the key down, the cursor continuously moves in the direction of the key you're pressing (saving you from having to perform the fabled "rapid-fire key-press").

INST/DEL

You can INSERT and DELETE letters and numbers from the line you are typing with this key. When you press this key by itself, the typed character immediately to the left of the cursor disappears, and the cursor moves over to where the missing character was. You can use the cursor keys to go back to the middle of a line and then use DEL to erase a letter. When you do this, the letter to the left is deleted, and the rest of the letters on the line move over one space to the left to close the gap.

To open up space to insert letters and numbers, type this key along with **SHIFT**. Space opens to the right of the cursor; the cursor itself does not move. When you insert space in the middle of a line of letters, the rest of the line moves to the right.

The **INST/DEL** key saves a lot of time when you want to edit or change what you've typed.

KEY	+		=	EFFECT
INST/DEL			=	DELETE (ERASE TYPED CHARACTERS)
SHIFT	+	**INST/DEL**	=	INSERT (ADD SPACES)

CLEAR/HOME

This key serves two primary functions: HOME and CLEAR. When you press this key alone, the cursor immediately moves to the top left corner of the screen (which is known as the HOME position). The rest of your screen stays the same. If you hold down the **SHIFT** key and press **CLEAR/HOME**, not only does the cursor move to HOME, but everything on the screen is erased (or cleared). All that remains on the screen is the blinking cursor at the top left corner of the screen.

KEY	+		=	EFFECT
CLEAR/HOME			=	HOME POSITION
CLEAR/HOME	+	**SHIFT**	=	CLEAR SCREEN

CTRL (Control)

The **CTRL** key is like the **SHIFT** key in that it always works with another key. You must hold it down while you press other keys. **CTRL** is used in three instances:

1) As the COLOUR KEYS section explains, pressing **CTRL** and a colour key lets you change the colour of the text printed on the screen.
2) You can pause a program that is PRINTing or LISTing on the screen by pressing **CTRL** and the S key.
   (To re-start the program, press any key except **RUN/STOP**.)
3) **CTRL** is also used with **REVERSE ON/OFF** and **FLASH ON/OFF**. These are explained later in this section.

In addition, some software programs that you buy make use of the **CTRL** key for their own special functions.

C= (COMMODORE KEY)

The Commodore key is very similar to CTRL, and can be used to perform four functions:

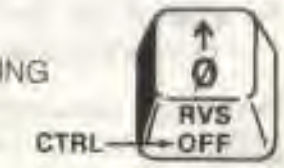
1) When used with the SHIFT key, the C= key lets you get into typing mode, where you can use both upper- and lowercase letters.
2) The C= key always acts as a shift to let you type the graphic symbol pictured on the LEFT front of each key. Just hold down C= and press the graphic key you want.
3) The C= key is used like the CTRL key to change the colour of what you type onscreen when pressed with a COLOUR KEY.
4) When you want to slow down a scrolling display (a program that appears line-by-line on the screen but might be going by too fast to follow), hold down the C= key. The display scrolling speed slows down considerably. When you release the key, it goes back to normal speed. (Hey, the C= key can do something by itself!)

RVS ON RVS OFF (Reverse On/Off)

Your Commodore 16 lets you print the reverse image of letters and numbers. In other words, if your cursor is black and the screen background is yellow, what you type appears in yellow letters on a black background.

Here's all you do to get reversed images: Press the CTRL key and the RVS ON key. Everything you type is now displayed in reverse, until you press the CTRL and RVS OFF, the RETURN key, or the ESC key and O. This returns you to typing normal (non-reversed) characters.

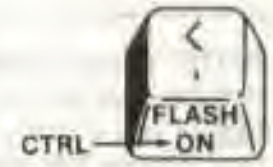
KEY	+		=	EFFECT
CTRL	+	RVS ON	=	REVERSE PRINTING ON
CTRL	+	RVS OFF	=	NORMAL PRINTING



FLASH ON FLASH OFF

You can make the characters on your screen flash on and off continuously, like the cursor flashes. Press CTRL and the FLASH ON key to make whatever you type flash. Typing CTRL and FLASH OFF, or RETURN, or ESC and O makes your typing normal (non-flashing) again.

KEY	+		=	EFFECT
CTRL	+	FLASH ON	=	CHARACTERS FLASH
CTRL	+	FLASH OFF	=	NORMAL DISPLAY



ESC (Escape)

The ESC key lets you perform many special screen editing functions, including functions to set up and manipulate screen windows. Windowing is a special property of your Commodore 16, which lets you set apart an area of the screen that may be used as work space without affecting the rest of the screen. The ESC key can perform several window editing functions, as well as many other regular uses, such as inserting, deleting, and scrolling. All the functions of the ESC key will be listed and explained when reviewing screen windows in Chapter 4.

Colour Keys

The colour keys bear a striking resemblance to the number keys from 1 to 8. When pressed with either the CTRL or the C= key, they change the screen colour of everything you type. Your Commodore 16 starts out with a white background and a blue border, with the cursor

flashing black. When you type anything, the characters appear in black. To change the cursor (and what you type) to a different colour, use the colour keys. On the front of the keys numbered from 1 to 8, there are two colours written. Press CTRL along with a number key to get the colour listed on top, or the C= key with a number to get the colour on the bottom. You can change only the colour of the characters using the colour keys. To change the screen background or border colour, you must use a BASIC command. (More on that later.)

NOTE: When changing colours, be careful that you do not press the number (colour) key even a split second before pressing CTRL or C=. If you do, all you'll do is print a number on the screen rather than change the colour.

KEY +	= EFFECT	KEY +	= EFFECT
CTRL + 1	= BLACK	C= + 1	= ORANGE
CTRL + 2	= WHITE	C= + 2	= BROWN
CTRL + 3	= RED	C= + 3	= YELLOW GREEN
CTRL + 4	= CYAN	C= + 4	= PINK
CTRL + 5	= PURPLE	C= + 5	= BLUE GREEN
CTRL + 6	= GREEN	C= + 6	= LIGHT BLUE
CTRL + 7	= BLUE	C= + 7	= DARK BLUE
CTRL + 8	= YELLOW	C= + 8	= LIGHT GREEN

Graphic Keys

Each letter key (and a few other symbol keys) on your Commodore 16 has two boxes on the front of the key, each with a different line, squiggle, or symbol. These are the graphic keys. When first turned on, your Commodore 16 prints capital letters. When you type one of these keys along with the SHIFT or C= key, you type the graphic symbols shown on the front of the key on the screen. You can type the full set of more than 60 graphics you see on the fronts of many of the keys.

KEY +	= EFFECT
SHIFT + A	= ♠ (GRAPHIC ON RIGHT SIDE OF KEY)
C= + A	= ┌ (GRAPHIC ON LEFT SIDE OF KEY)

You can create pictures, charts, and designs by printing graphics side by side or on top of each other, like building blocks. You can make your graphics more interesting by using the colour keys to "draw" in different colours. Try printing with some of the graphics keys to see how they work. Chapter 5 explains more about graphics.

When you're in typing mode, you can only use the graphic symbols on the left front of the keys (by pressing C= and the appropriate key). The left side graphics are ideal for creating charts, graphs, and business forms.

Function Keys

The four keys on the right side of your keyboard (apart from the rest of the keys) are special function keys that let you save time by performing repetitive tasks with the stroke of just one key. The top of each key reads f1, f2, f3, and HELP. You can get these functions by just pressing the key by itself. The fronts of the keys read f4, f5, f6 and f7. Press SHIFT and f1, f2, f3 and HELP, respectively, to get these functions.

Here's what each key does:

KEY 1 enters one of the GRAPHICS modes when you supply the number of the graphics area and press RETURN. The GRAPHICS command is necessary for giving graphics commands such as CIRCLE or PAINT. More on GRAPHICS in Chapter 6.
KEY 2 prints **DLOAD"** on the screen. All you do is enter the program name to load a program from disk and hit RETURN instead of typing out **DLOAD"** yourself.
KEY 3 lists a DIRECTORY of files on the disk in the disk drive.
KEY 4 clears the screen using the SCNCLR command.
KEY 5 prints **DSAVE"** on the screen. All you do is enter the program name to save the current program on disk and press RETURN.
KEY 6 RUNs the current program.
KEY 7 displays a LISTing of the current program.
KEY 8 (the HELP key) highlights errors in program statements in flashing print.

You can redefine any of these keys to perform a function that suits your needs. Redefining is easy, using the KEY command. You can redefine the keys from BASIC programs, or change them at any time in direct mode. A situation where you might want to redefine a function key is when you use a command frequently, and want to save time instead of repeatedly typing in the command. The new definitions are erased when you turn off your computer. You can redefine as many keys as you want and as many times as you want.

The Help Key

When you make an error in a program, your computer displays an error message to tell you what you did wrong. These error messages are further explained in the Appendices in the back of this manual.

You can get more assistance with errors by using the HELP key. After an error message, press HELP to locate your error exactly. When you press HELP, the line with the error is displayed on the screen with the error flashing on and off. For example:

```
?SYNTAX ERROR IN LINE 10
HELP
10 PRONT "COMMODORE
COMPUTERS"
```

Your computer displays this
You press HELP

The mistake is displayed flashing on and off.

CHAPTER 3

SOFTWARE

- Introduction
- Cartridges
- Cassettes
- Diskettes

INTRODUCTION

A computer without software is like a glass without orange juice. Well, maybe not, but software does add to the usefulness of your computer and the fun and diversion it affords you. Software is all the programs that can be entered and run on a computer. The hardware (in this case, your Commodore 16) can use software in many forms: plug-in cartridges, pre-recorded tapes and diskettes. The family of software available for your Commodore 16 is growing quickly. Your dealer can keep you up-to-date on new products and inform you about the features of software that's currently available.

Your Commodore 16 can use software on CARTRIDGE, CASSETTE TAPE, and DISKETTE form, available from your Commodore dealer. All you do is load them into your Commodore 16. You can also create and store your own programs on cassette tapes or floppy disks.

CARTRIDGES

Commodore produces a full assortment of cartridge software for your Commodore 16. There is a variety of personal, education, and business programs, as well as exciting games available for your Commodore 16. You don't need any additional equipment to use cartridge software; all you do is plug the cartridge into the back of your computer and power on. Here are the steps to follow to use cartridges:

Loading Cartridges

STEP 1 Turn OFF your Commodore 16.

IMPORTANT: YOU MUST **TURN OFF** YOUR COMPUTER BEFORE YOU **INSERT OR REMOVE** CARTRIDGES. IF YOU DON'T, YOU MAY **DAMAGE** THE **CARTRIDGE** AND THE **COMPUTER**.

STEP 2 Hold the cartridge with the label facing UP, and insert the cartridge firmly in the cartridge slot (labelled 'memory expansion') in the back of your computer.

STEP 3 Turn ON your Commodore 16.

STEP 4 Begin the game or program according to the instructions that come with the software.

CASSETTES

A variety of software for the Commodore 16 is available on cassette tape. These cassette tapes are similar to the music cassettes that you play on your tape deck or stereo. Software on tape works in the same fashion as cartridge software, but you have to have an additional piece of equipment (called a peripheral) to load cassette software into your computer. To use cassettes, you need a Datassette tape recorder, available from your Commodore dealer.

You can also use cassette tapes and the Datassette to store programs you write yourself. The next section explains how to save programs on tape.

The steps for loading a program on cassette tape are the same, whether you're using pre-recorded software or programs you saved yourself.

Loading Cassette Tapes

STEP 1 Insert the cassette into your Datassette and close the door.

STEP 2 Rewind the tape to the beginning by pressing the REWIND button on the Datassette.

STEP 3 Press STOP when the tape is rewound to the beginning, type **LOAD** and press the **RETURN** key. The computer responds with the following message:

PRESS PLAY ON TAPE

STEP 4 Press the PLAY button on the Datassette. The screen goes blank as the Datassette starts. When a program is found, the screen displays this message:

SEARCHING

FOUND program name

STEP 5 Press the Commodore key **C=** to load the program that was FOUND. If there is more than one program on the tape, and the program your Commodore 16 found isn't the one you want, do nothing. Your computer will continue searching after a brief interval.

When the program is loaded, the word READY appears. If you want to stop the loading before it's complete, press **RUN/STOP** on the keyboard, and then the STOP button on the Datassette. After the software is loaded, type **RUN** to start the program. You can also LIST the program or change it, if it is a BASIC program.

Loading A Specific Program

To LOAD a specific program on the tape, use the **LOAD** "program name" form of the LOAD command. The instructions are the same as typing **LOAD** with no name, with just a few differences.

STEP 1 If the program you want is called "BASES", you would type:

LOAD "BASES"

and press **RETURN**.

Your computer responds with:

PRESS PLAY ON TAPE

STEP 2 Press the PLAY button on your Datassette to get your computer to start looking. After searching on the tape for the program called BASES, the message should appear:

FOUND BASES
LOADING

The screen goes blank while your computer then "reads" the program into its memory. If the entire tape goes by without the FOUND message, rewind the tape and try again. Once your computer has digested the entire program, your Commodore 16 tells you:

READY.
■

STEP 3 At the cursor, you instruct it to

RUN

and press **RETURN**. At this point, your Commodore 16 runs (executes) the program "BASES".

Saving Programs on Cassette Tape

When you write a program yourself and you want to keep it for later use or modification, you can SAVE that program. When you SAVE a program, you are in effect recording it on a form of software (tape or disk) that allows you to recall the program so you can RUN it, make changes, etc. When you want to save a program on cassette tape, follow these steps:

STEP 1 Type:

SAVE "program name"

The program name you use can be anything you want, but can be no more than 16 letters and/or numbers long.

STEP 2 Press the **RETURN** key. The computer displays this message:

PRESS PLAY & RECORD ON TAPE

STEP 3 Press the RECORD and PLAY buttons on your Datassette. The screen goes blank. When your program is saved, the word **READY** appears on the screen.

Examples of SAVE Commands for Cassette Tape:

SAVE "THE DAY"	This name is the specific name
SAVE "YOURSELF"	of the program being saved

NOTE: When saving a program onto a cassette tape, always be aware of where the tape is positioned.

DISKETTES

Disks are fast and easy to use. Be sure to handle your disks and your disk drive carefully. Disks may be referred to as diskettes, floppy disks, or floppies interchangeably; they are all the same thing. Unlike

cassettes, you only have to put the disk into the disk drive and type the commands to LOAD or SAVE programs; there are no buttons to push. There are a couple of small lights on the front of the disk drive. The green light is the power light, telling you whether the disk drive is turned on or off. The red light tells you two things. During normal disk drive use, when a program is being LOADed or SAVEd, it is lit while the disk is spinning in the drive. If there is a problem with the diskette or drive, the red light flashes on and off, even after the disk stops spinning in the drive.

Loading Programs From Diskette

STEP 1 Make sure that your disk drive is plugged in and the serial cable is connected. Then turn the power ON.

STEP 2 Insert the disk into the disk drive. The label side of the disk must face up. Insert the disk into the opening so that the labelled end goes in last. Look for a little notch on the side of the disk (it might be covered with a sticker). This notch should be to your left as you put in the disk, assuming that you're facing your disk drive. Be sure the disk is in all the way.

STEP 3 Close the protective "door" on the disk drive after you insert the disk.

STEP 4 Type:

DLOAD "program name" ← Specific name of the program to be LOADed.

(To save time, you could press FUNCTION KEY 2 and type in the program name and the second quote marks.)

STEP 5 Press the **RETURN** key. The disk spins and your screen says:

SEARCHING FOR program name
FOUND program name
LOADING

READY.
■ ← cursor

STEP 6 Your software is now ready to use. Type **RUN** and press the **RETURN** key to start the program.

If the red light on the disk drive blinks after the DLOAD is finished, something went wrong. Type:

?DS$ (and hit **RETURN**)

to find out what went wrong.

Examples of DLOAD commands:

DLOAD "*"	LOADs the 1st program on the disk.
DLOAD "**FILES**"	LOADs a disk program called FILES.
DLOAD "**SOF***"	LOADs the first program on the disk that begins with the letters SOF.

Headering A Diskette

Headering prepares a new BLANK disk for use. Any blank disk must be formatted before it may be used, by using the HEADER command.

IMPORTANT: DO NOT HEADER A DISK THAT HAS ANYTHING ON IT UNLESS YOU WANT TO ERASE THE ENTIRE DISK. **HEADERING ERASES** EVERYTHING ALREADY ON A DISK.

The format for the HEADER command is:

HEADER "disk name" [,**U**device#] [,**I**i.d.#],**D**drive#

- The name you use is the name of the entire disk. Give the disk any name up to 16 characters.
- **U**device # specifies which peripheral device number (which disk drive unit), and defaults to 8 (primary disk drive).
- The i.d. is the letter **I** and any two letters and/or numbers, like **I**21, **I**R5, etc. Give the disk any i.d. you want, but you should give every disk a different i.d. to avoid confusion. (Note: the first character must not be **F**).
- If you have a dual drive (a disk drive with two slots for disks), you should specify the drive number with a 0 or 1. Drive number is ALWAYS included in the command, even with a single drive (**D**0).

ARE YOU SURE?
As soon as you press **RETURN** after typing the HEADER command, your Commodore 16 asks ARE YOU SURE? This is to give you a last chance to change your mind. If you want to go ahead and header the disk, type **YES** or **Y** and press **RETURN**. If you decide not to header the disk, type **NO** or **N** and press **RETURN**.

Here are some examples of HEADER commands:

HEADER "LETTERS",I07,D1
HEADER "FINANCES",U8,IS3,D0

NOTE: If you want to erase all files on a disk (as opposed to formatting a blank disk), use the HEADER command without an i.d. number. This in effect scratches (erases) all the files that were on that disk.

Now that you know how to HEADER a disk, you're ready to use disks to write and save programs on your Commodore 16 (or any other

Commodore computer). The BASIC Encyclopedia in the back of this manual has more information about the HEADER command.

Saving Programs on Diskette

When you want to reuse a program you've written, be sure to SAVE it before you LOAD another program or turn off the Commodore 16. If you don't, you'll lose the program.

When you change a SAVEd program, you have to SAVE it again if you want to keep the new version.

When you reSAVE a program, you are replacing the old version with the new one. If you want to keep both the old and the changed versions, you have to give the new one a different name when you SAVE it.

Follow these steps to save a program on disk:

STEP 1 Type **DSAVE** "program name"

STEP 2 Press **RETURN**. The computer displays this message when the program is saved:

```
SAVING 0: "program name"
READY.
```

Example:

```
DSAVE "MYPROGS"
```
The program name can be up to 16 characters long.

If the red light on the disk drive blinks after the DSAVE is finished, something went wrong. Type:

```
?DS$ (and hit RETURN )
```

to learn what went wrong.

If you try to SAVE a program on a write-protected disk (a disk that does not accept your output), you must turn the drive OFF, then ON again.

The Directory Command

When you SAVE programs on disk, the computer keeps a listing of all the files saved on that disk. You can display the listing as a table of contents to see what's on a disk by using the directory command.

Type:

```
DIRECTORY
```
then press **RETURN**
(or press FUNCTION KEY 3)

As soon as you press **RETURN**, your Commodore 16 displays everything on your disk.

You can also display just part of the table of contents:

```
DIRECTORY "MY*"
```
Lists every file on the disk that starts with the letters MY.

```
DIRECTORY "*=PRG"
```
Lists every program file.

```
DIRECTORY "*=SEQ"
```
Lists every sequential file.

CHAPTER 4

THE FIRST STEPS

- Introduction
- The screen display
- Reverse printing and changing colours
- The first step
- Correcting mistakes
- Screen windows

INTRODUCTION

The purpose of this chapter is to begin to acquaint you with some of the characteristics and capabilities of your Commodore 16, and how to take the first steps toward programming with your computer.

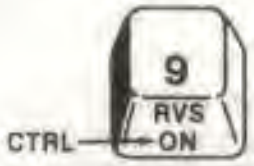
The Screen Display

Your Commodore 16's comings and goings are seen on your TV or monitor display screen. Whatever you type, whatever your computer has to say about what you type, whatever programs you load . . . it all shows up on your monitor screen, whether you're hooked up to a colour or black and white TV, or a Commodore colour monitor. Your Commodore 16 screen, no matter what type of monitor or TV you're using, is 40 columns wide. That means that you can type 40 characters from the left edge of the screen to the right. There are 25 rows, so that an entirely filled screen can contain spaces for 1000 (25 × 40) characters. The Commodore 16's screen display is made up of three elements: the characters (the letters, numbers and graphic symbols), the background, and the border. When you first turn your computer on, the characters are black, the background is white, and the border is light blue. You've already seen how to change the colour of the characters using the number keys along with **CTRL** or **C=** in Chapter 2. You can make the characters any of 16 different colours in this fashion. There are a few other things you can do to make printing characters more interesting.

Reverse Printing and Changing Colours

Not only can you change the colour of the cursor and characters you type (and your computer displays) on your monitor screen, you can also reverse the character and background colours. That means that if you have black characters on a white background and you turn reverse printing on (by typing **CTRL** and 9), everything that appears on the screen is reversed: the characters are in white, while the background behind those characters is black. You can experiment with changing the character colour and using reverse printing, by following these instructions:

STEP 1 Hold down the **CTRL** key and press the **RVS ON** key:



STEP 2 Release the keys and hold down the space bar.

STEP 3 Hold down the space bar as long as you want. While you hold down the space bar, a line the same colour as the letters on your screen should get longer. If the line gets to the end of the row, it continues on the next row.

STEP 4 Release the space bar (but don't press the **RETURN** key).

STEP 5 Hold down the **CTRL** key and press one of the colour keys (not a colour that's already on your screen). As soon as you do this, the cursor changes to the colour of the key you pressed.

STEP 6 Hold the space bar down again. Now your computer draws a line in the new colour. Keep changing colours with the **CTRL** or **C=** keys and the colour keys. Then hold down the space bar to make more different coloured lines.

STEP 7 Turn off reverse print by holding down **CTRL** and pressing the **RVS OFF** key. Pressing the **RETURN** key also turns off reverse printing.

BRIEF INTERLUDE: You can type letters, numbers and punctuation in reversed print. Reverse letters make excellent headlines. You can also use them to highlight special words and numbers. You can type graphic symbols in reverse for the same effect. To put a racing stripe on your screen, select a colour you like and type the symbol — (by pressing SHIFT and E) across an entire line. Then turn on reverse printing; fill the next line with the same symbol, only reversed. On the next line, type SHIFT and R (still in reversed print). Then turn off reverse, and fill the last line with SHIFT and R (no longer reversed). Now your screen has a racing stripe in whatever colour you chose; perhaps it is a humble acheivement, but it does give your screen a certain flair . . .

THE FIRST STEP

What is a program, anyway? An experienced programmer might tell you that a program is a series of ordered statements organised to direct the computer to perform a sequence of steps toward some purposeful result. What you may learn from this is not to ask an experienced programmer what a program is.

The following, although it is only two lines long, qualifies as a program. Type it exactly as it appears here. Don't leave out the numbers at the beginning of each line, since they are the line numbers; they tell your computer what order to read and perform the lines of the program. Be sure to press the RETURN key at the end of each line you type.

10 PRINT"FIRST TRY"	This line tells your computer to print the words FIRST TRY on your screen.
20 GOTO 10	This line tells your computer to go back to line 10 and print FIRST TRY again.
RUN	When you type this and press RETURN, your computer does what each line tells it to do.

Press the RUN/STOP key to stop the program.

Why did your computer print FIRST TRY so many times? PRINT is a **command** that tells your computer to PRINT what appears between the quote marks on the screen. When your computer reads this line, it

executes (carries out) that command. GOTO tells your computer to go back to line 10 and execute the command (PRINT "FIRST TRY"). Each time your computer got to line 20, it executed the GOTO command, returning to line 10 again and again. This repetition is called a loop. The RUN/STOP key breaks into whatever program the computer is running, and returns control of the keyboard back to you.

TIP: Want to slow down this program (or any other) without stopping it? Just hold down the C= key.

Now type the following, pressing RETURN after each entry:

NEW ← This tells the computer to forget the last program and get ready for a new one.

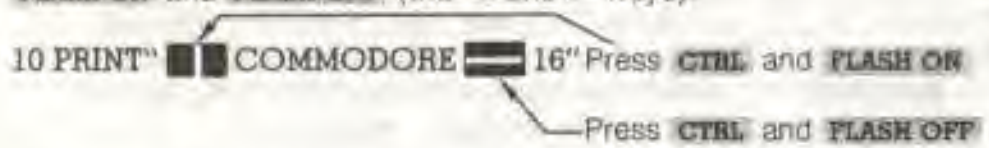
The computer responds with the message:

READY ← You don't type this; this is your computer letting you know that it's READY for new commands or programs.

10 PRINT " R COMMODORE — 16" Press CTRL and RVS ON
Press CTRL and RVS OFF

Notice what happens when you type RVS ON and RVS OFF inside the quote marks. They appear on the screen as reversed characters. Now type RUN and press RETURN to see the program. When the program is RUN, your Commodore 16 reads these reverse characters as what you typed in, and carries out the reverse print as instructed. The same principle holds true (a reversed character appearing on the screen as a representation for what you type) for doing other things in a PRINT statement, like moving the cursor and changing colours, when you RUN a program.

Now try the same line, but replace **RVS ON** and **RVS OFF** with **FLASH ON** and **FLASH OFF** (the < and > keys):



When you RUN this, it PRINTS out COMMODORE 16 just once. But the word COMMODORE flashes continuously, even though your program has already been RUN. When you use flashing print, it will flash continuously. Both of these one-line programs show how to use things like reverse printing and flashing letters in the line of a program. You would do other things (change colours, etc.) in the same manner.

Entering Commands

You might have noticed that you typed some things in as just a single word (such as **NEW**), while on other lines you had to type in line numbers followed by commands and statements within quote marks. That's because there are two 'modes' that you can use to communicate with your computer. Both are based on a language made up of terms that your computer understands. The most straight-forward (and the one built in to your Commodore 16) is called BASIC. Your Commodore 16 understands a version called Commodore BASIC 3.5. The BASIC terms (or keywords) are the heart of both modes. The first type, IMMEDIATE MODE, tells your computer to execute the BASIC command immediately. You enter the command to be carried out by your computer when you press the **RETURN** key. This is also known as DIRECT MODE. The alternate mode is known as PROGRAMMING, or INDIRECT MODE. Programming mode features line numbers, and each line contains BASIC commands. The entire program is executed when you type the command **RUN**, carrying out the commands of the lines according to line number (lowest lines first). Obviously, programming mode is what you use to write computer programs.

CORRECTING MISTAKES

Mistakes are a way of life with computers. Mistakes in programming must be searched for and remedied for a program to run correctly. That's what the **HELP** key is for. A more practical kind of mistake—the typing error—can mess up programs just as badly. Not only will computing improve your typing skills, you'll make great strides in spotting and correcting typing errors as well. There are several ways to fix up typing errors. Remember that to enter any changes (or anything, for that matter) either for your computer to execute or into your computer's memory, you must press the **RETURN** key.

1. YOU CAN EDIT A LINE by overtyping.
Use the CURSOR KEYS to move to the place in the line that you want to change. Now just type over what you want to change. Press **RETURN** when you finish.

EXAMPLE:

```
10 PRINT "IT IS ONE O'CLOCK"
```

If you want to change the time to TWO, move the cursor to the O in ONE.

```
10 PRINT "IT IS ONE O'CLOCK"
```

And now just type TWO over ONE and press **RETURN**.

```
10 PRINT "IT IS TWO O'CLOCK"
```

> **NOTE:** When working with numbered program lines, you don't have to be at the end of the line to press **RETURN**. Your Commodore 16 remembers the whole line even if you press **RETURN** in the middle of the line.

2. YOU CAN OPEN UP SPACES IN A WORD OR LINE with the **INST** key (press **SHIFT** along with the **INST/DEL** key). Hold the keys down until you open up as many spaces as you need. (Notice that the cursor stays in the same place while spaces open up to the right.) Then just type what you want to insert.

```
10 PRINT "CORE"
```

To change this to COMMODORE, move the cursor to the 'O' and press the **SHIFT** and **INST** keys until enough space opens up. Don't bother to count out the spaces. You can just guess and then open up more if there aren't enough.

```
10 PRINT "C■      ORE"      cursor
```

Now add the other letters:

```
10 PRINT "COMMODORE"
```

3. **YOU CAN ERASE CHARACTERS AND CLOSE UP SPACE** with the **DEL** key (get delete by pressing the **INST/DEL** key by itself). This key erases characters or spaces immediately to the LEFT of the cursor.

```
10 PRINT "AFTERNOON SCHEDULE"
```

You can change this to WEEKLY SCHEDULE by moving the cursor to the E in AFTERNOON, pressing the **INST/DEL** key three times, and typing **WEEKLY**.

```
10 PRINT "AFT E RNOON SCHEDULE"
```

and press **INST/DEL** three times.

```
10 PRINT "  E RNOON SCHEDULE"
```

Type in **WEEKLY** to replace **ERNOON** and press **RETURN**

4. **YOU CAN RETYPE A LINE** anytime, even after you've RUN the program. Your Commodore 16 automatically replaces the old line with the new one when you press **RETURN** to enter the new line. The old line still appears on the screen, but your computer ignores it. When you have two statements with the same line number, your Commodore 16 only uses the last one entered. For example, in a brief program using the COLOR command to change the colour of the screen background, a mistake might occur:

```
10 COKOR 0,3                    mistake
20 PRINT "COMMODORE 16"
```

Press the **RETURN** key to get to a fresh line, and just retype line 10 correctly:

```
10 COLOR 0,3
```

Now the first line 10 is replaced by the second line 10. You can check

this by typing **LIST**, which displays a line-by-line LISTing of your program as it is stored in the computer's memory. When you LIST a program, all lines appear in correct order and the replaced lines don't appear.

```
LIST                    RETURN
```

The screen reads:

```
10 COLOR 0,3
20 PRINT "COMMODORE 16"
```

You can see what the COLOR command does in this example. For a more complete explanation, see Chapter 6, or look up COLOR in the BASIC Encyclopedia.

Replacing lines in a program is also a good way to experiment with your computer. When you replace a line, the new one doesn't have to be anything like the old line. For example, instead of correcting the spelling of COLOR, you can enter this line:

```
10 PRINT"COMMODORE 4 TIMES COMMODORE 4 = "
                                          \
                                           space
```

Now RUN the program and see what happens.

5. **YOU CAN ERASE AN UNWANTED LINE** just by typing the **number** of the line and pressing **RETURN**. The computer ignores the line even though it still appears on the screen. Type **LIST** to get the program LISTing to make sure the line is gone from the program.

```
10 PRINT "COMMODORE 4 TIMES COMMODORE 4 = "
20 PRINT "COMMODORE 16"
```

Now type:

```
10                              RETURN
```

```
LIST
```

The result, according to your computer:

```
20 PRINT "COMMODORE 16"
```

CLEARING THE SCREEN

There will be times when your screen is too crowded, or you want to reorganize what you've typed, or you're just plain unhappy with what's on the screen and want to banish it to wherever it is that print goes once it leaves the monitor screen. There are a few ways to accomplish this, leaving the programs in your computer's memory intact or clearing the memory as well as the screen.

One way to clear the screen is to hold the **SPACE BAR** down, until it clears the 1000 spaces on the screen (for a couple minutes or so). If you've read this far into the User's Manual, chances are that you have the patience to clear the screen like this, but also the intelligence to realise that there has to be a better way. Here are some better ways:

1. Hold the DOWN CURSOR key ↓ until everything on the screen scrolls off the top of the screen.
2. Press the **SHIFT** and **CLR/HOME** keys together. This clears the screen and repositions the cursor at the top left-hand corner (the 'home position')
3. Type:

   SCNCLR and press **RETURN**

   This is a BASIC language command that your computer interprets as instruction to clear the screen.
4. Press the RESET BUTTON. This resets the machine, bringing back the starting screen and clearing all programs out of your computer's memory.

If you want to clear the computer's memory but not the screen, type:

NEW and press **RETURN**

This is a BASIC language command that tells the computer to empty its memory. The 16 in the Commodore 16 refers to the amount of space (16K) that your computer has to store programs, known as the memory. When you clear the screen with any of the first three methods, you clear the characters on the screen, but whatever is in your computer's memory stays there. This means you have less space for new programs. The last two methods erase what is held in your computer's memory, so that you again have the full amount of space available for new programs.

SCREEN WINDOWS

Windows let you define a specific area of the screen as your workspace. Everything you type (lines you type, LISTings of programs, etc.) after setting a window appears within the window's boundaries, not affecting the screen outside the window area. You can set up a window anywhere on the screen.

To set a window, follow these steps:

STEP 1 Move the cursor to the screen position you want as the top left corner of the window.

STEP 2 Press the **ESC** key and release it, and then press 'T'.

STEP 3 Move the cursor to the position you want to be the bottom right corner of the window.

STEP 4 Press **ESC** and release, then 'B'. Your window is now set.

All screen output is confined to the 'box' you have defined. To cancel the window, press the **CLEAR/HOME** key twice. The window is then erased, and the cursor is positioned in the top left corner of the screen.

You can manipulate the window and the text inside using the **ESC** key. Screen editing functions, such as inserting and deleting text, scrolling, and changing the size of the window, can be performed by pressing **ESC** followed by another key. To use a specific function, press the appropriate key after pressing **ESC**.

KEY	FUNCTION
A	Automatic insert
B	Set the bottom right corner of the screen window (at the current cursor location)
C	Cancel automatic insert
D	Delete current line
I	Insert a line
J	Move to the beginning of the current line
K	Move to the end of the current line
L	Turn on scrolling
M	Turn off scrolling
N	Return to normal screen display size
O	Cancel insert, quote, reverse, and flash modes
P	Erase everything from the beginning of the line to the cursor
Q	Erase everything from the cursor to the end of the line
R	Reduce screen display
T	Set the top left corner of the screen window
V	Scroll screen up
W	Scroll screen down
X	Cancels the previous Escape function

CHAPTER 5

NUMBERS AND CALCULATIONS

- Introduction
- Numbers and operations
- Calculations
- More about screen PRINTing
- Variables
- Numeric functions
- User defined functions

INTRODUCTION

You don't have to be a mathematics genius to understand and make use of the mathematics capabilities of your Commodore 16. In addition to the straight-forward operations like addition, subtraction, multiplication and division, you can use your computer to figure out advanced functions like square roots and sines. You'll learn about the different types of variables, and how to use them. Your computer can handle single digit whole numbers or complex numbers of up to 38 places expressed in exponential notation with equal efficiency. You can do your calculations directly or as part of programs. Finally, this chapter gives you a brief explanation of how to set up your own functions for your computer to evaluate.

NUMBERS AND OPERATIONS

You can use your computer like a simple calculator. Besides the standard + and − operation signs, your Commodore 16 uses the * sign for multiplication and the / sign for division and fractions. (Computers use the * sign instead of an X for multiplication because a computer can't tell the difference between the letter X and the mathematical symbol X.) You can use these operators and numbers in immediate mode (no line numbers) or in program lines. Type in numbers and operators in PRINT statements WITHOUT QUOTES if you want your computer to perform the math involved. If the numbers and operators are in quotes (as in PRINT "2 + 2"), your computer will PRINT exactly that on the screen, instead of performing the addition. You can get your computer to put two and two together by typing PRINT 2 + 2.

BASIC MATHEMATICAL OPERATORS		BASIC RELATIONAL OPERATORS	
Addition	+	Greater than	>
Subtraction	−	Less than	<
Division and fractions	/	Equals	=
Multiplication	*	Greater than or equal	=> or >=
Exponentiation (press SHIFT and 0)	↑	Less than or equal	<= or =<
		Not equal to	< > or > <

NOTE: Your computer doesn't accept commas as part of a number. For example, you have to type 30359 instead of 30,359. If you put a comma in a number, your computer thinks you mean two numbers (separated by the comma), and would read it as 30 and 359 instead of 30359.

Fractions and Decimals

You can write a fraction like this: 5
or like this: 1/2 Your computer is actually performing the division

If you put a fraction in a PRINT statement, your answer is always returned as a decimal or whole number. For example:

```
PRINT 139/493 + 5                    RETURN
 5.28194728
```

π THE PI KEY

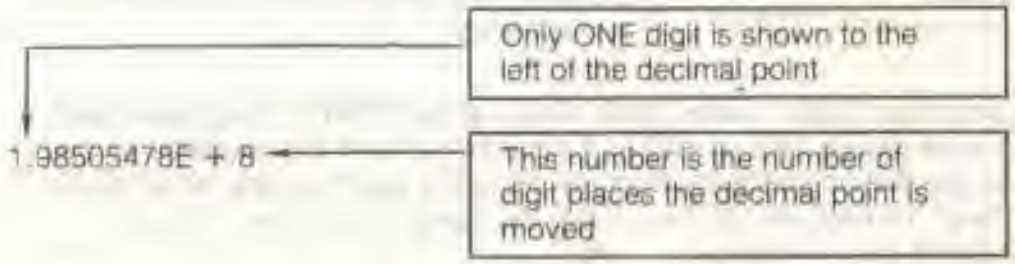
Here's an example that uses pi (3.14159265 . . .), which represents the ratio of the circumference of a circle to its diameter. Use this value by just pressing the π key (C= and =):

```
PRINT π /374                         RETURN
 8.39998036E-03
```

Scientific Notation

What did your computer mean by the E-03 part of the above answer? Your computer displays decimal numbers in the range -999,999,999 to 999,999,999 in standard numerals. Numbers beyond this range (with more than nine digits) are automatically displayed in scientific notation. You can enter numbers in yourself in this form and your computer will read them with no trouble (certainly less trouble than you had converting them!). Scientific notation is often useful, since this special notation lets your computer display large numbers in fewer digits.

Here is how the number 198,505,478 would be written in scientific notation:



For a number less than one with several decimal places, the second number would be a − instead of a +, indicating that the decimal point is moved to the right.

For example:

.0003359 = 3.359E − 4

Other examples:

20 = 2E + 1	the decimal point is moved 1 digit left
105000 = 1.05E + 5	the decimal point is moved 5 digits left
.0666 = 6.66E − 2	the decimal point is moved 2 digits right

PERFORMING CALCULATIONS

To perform a calculation, type PRINT and then the math problem without quotes, as in the following:

```
10 PRINT 1+2, 2-1
20 PRINT 2*2, 4/2
RUN
```

3 1

4 2

For the first time, PRINT didn't print exactly what you typed in the statement. Instead, your computer solved the calculations and PRINTed only the answers. All you have to do to use PRINT to calculate is omit the quotation marks. Now try this:

```
NEW
10 PRINT "2001/2010"
20 PRINT 2001 - 17
RUN
2001/2010
 1984
```

one space is left here for the answer's sign

Since line 10 is in quotes, your computer just PRINTs the problem as if it were regular text: exactly as it appears between the quotation marks. Notice that no space is left for the number's sign from line 10, as it is in the printout for line 20. Now move the cursor back to line 10 and change the line to this:


```
10 PRINT "2001/2010=";
   2001/2010
RUN
2001/2010= .995522388
 1984
```

don't forget the semicolon

this space is left for the answer's sign

the answer for line 20 stays the same

If you want to both PRINT the problem AND solve it you have to type it twice: once in quotes and once out of quotes, as line 10 shows.

Order of Calculation

You can perform more than one calculation in one line. Try typing this:

```
PRINT 200*50+5
```

Is the answer what you expected? Try this:

```
PRINT 50+5*200
```

Your Commodore 16 always performs calculations in a certain order. Problems are solved from left to right; within that general rule, some types of calculations are solved first. The order that your computer evaluates expressions is called the order of precedence.

FIRST: Your computer checks for negative numbers (not subtraction, just negative numbers).
SECOND: Your computer solves any exponents.
THIRD: Your computer solves all multiplication and division, from left to right.
FOURTH: Your computer solves addition and subtraction, from left to right.

NOTE: Your Commodore 16 always solves any part of the problem that's surrounded by parentheses first. You can even put parentheses within parentheses: 36 * (12 + (A / 3)). The contents of the innermost parentheses are solved first.

Sometimes it's a good idea to put negative numbers in parentheses for clarity. For example, if you want to multiply 45 by -5, type it like this: 45*(−5). Your computer can understand it with or without parentheses.

MORE ABOUT PRINTING ON THE SCREEN

You've probably noticed that certain programs had you type commas in certain places, while in others there were semicolons. Your computer interprets commas and semicolons as instructions for the spacing of the printout.

The effect of punctuation on the spacing of PRINT statements works the same with spacing of both text in quotes (called "text strings") and numbers being calculated. Try typing this brief program:

```
NEW
10 PRINT "O","K"
20 PRINT "O";"K"
```

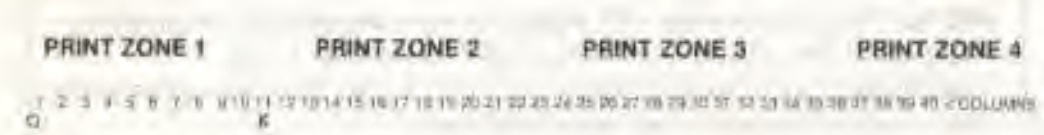
Notice that the punctuation appears OUTSIDE of the quote marks. When you RUN the program, the screen looks like this:

```
O       K                    line 10 PRINTed this
OK                           line 20 PRINTed this
```

If line 10 and line 20 are nearly identical, why is there such a difference in what they PRINT on the screen? The only difference is due to the punctuation between the items this program PRINTs.

When you use a comma to separate items in a PRINT statement, the items are PRINTed several spaces apart. When you use a semicolon, the items are PRINTed right next to each other.

As you recall, your computer's screen has 40 columns across. These columns are divided into four 10 space areas, called PRINT ZONES. When you use a comma to separate PRINTed items, your Commodore 16 PRINTs the first item in the first print zone, the second item in the second print zone, etc. The commas work like tabs on a typewriter.



If you try to PRINT more than four items separated by commas, your computer automatically goes to the next line to PRINT. For example:


```
PRINT "A","B","C","D","E","F"
```

spaces the letters like this on your screen:

```
                1          11         21         31         40    COLUMN
ROW 1           A          B          C          D
    2           E          F
```

When you use semicolons to separate items in a PRINT statement, your Commodore 16 ignores the print zones and PRINTs all the items one after another:

```
PRINT "A";"B";"C";"D";"E";"F"
```

PRINTs this:

```
ABCDEF
```

If the first PRINT item is 12 letters long and the second item is separated by a comma, here's what happens:

```
PRINT "ABCDEFGHIJKL","M"
```

PRINTs this:

```
ZONE 1      ZONE 2      ZONE 3
ABCDEFGHIJKL            M
```

NOTE: Sometimes you'll type a particularly long line on your computer, such as this:

```
10 PRINT "I LIKE YOUR TOUCH ON MY KEYBOARD. DO YOU
COME HERE OFTEN?"
```

You'll notice that as you type this, you run out of room on one row. But keep typing; the Commodore 16 automatically moves on to the next row and continues printing there until your line is finished. You can type as many as 80 characters on one program line (up to two full 40-column rows)

Now try RUNning this one line program. The message is printed on two

rows. If your line is longer than one row, your Commodore 16 lets it spill over to the next row. Your computer considers the line ended when you press the RETURN key, not when you type to the end of the row. You'll get used to this as you use your Commodore 16.

USING VARIABLES

The example 36*(12+(A/3)) shows one of the most powerful features of a computer. When you have a letter instead of a number in a mathematical problem, you're using a VARIABLE. A variable represents a value:

```
10 A = 3
20 PRINT "TOTAL:";A * 4
```

If you RUN this program, the screen result is:

```
TOTAL: 12
```

There are three types of variables you can use:

TYPE	SYMBOL	DESCRIPTION	EXAMPLES	SAMPLE VALUES
Floating point		real (decimal) or whole numbers	X, AB, T4	23.5, 12, 1.3E+2
Integer	%	whole numbers	X%, A1%	15, 102, 3
Text string	$	letters, numbers, and all other characters in quotes	X$, MS$	"TOTAL:", "DAY 1", "CBM"

Every time you want a variable to be read as a whole number, the symbol for that variable would include the % sign. A variable that contains text MUST end with a $ as part of the variable. If it doesn't have that symbol, your computer expects a floating point number. A variable without either of the symbols (% or $) is read as a floating point number (a "regular" number). Integer variables (whole numbers) are a subset of floating point variables; they are numbers with no decimal places.

Make sure that you always use the right variable type. If you try to do something like assign a word to an integer variable, your program won't work. This program shows you what variable can or can't be used in a given situation, and you can find out what happens when you try out different types of data:

```
10 PRINT "ENTER A NUMBER"

20 INPUT X%

30 PRINT "I READ YOUR NUMBER AS"; X%

40 PRINT "NICE GOING, ACE!"

50 END
```

this asks you to enter a number for the variable

When you RUN this program, try to enter these values (one each time you RUN it) when you're asked and see what happens:

ONE FIFTH
.043
10

NUMERIC FUNCTIONS

Included in your computer's BASIC language are numeric functions, which are like the advanced calculations found on most scientific calculators (such as sine, cosine, tangent, etc.)

Most of the functions can be used by typing the name of the function and the number to be operated by the formula in parentheses, like this:

```
FUNCTION(X)
```

For example, to find out the sine of a variable, you would type:

```
PRINT SIN(X)
```

with X as any number you want to input.

You could also include one of the functions in a program line, as the following example shows:

```
10 FOR X= 1 TO 5
20 PRINT "THE SQUARE ROOT OF"; X;"IS"; SQR(X)
30 NEXT X
```

You'll find a complete listing of the numeric functions in the BASIC Encyclopedia in the back of this manual.

NOTE: Most commands have an abbreviation that you can type instead of typing out the entire BASIC name. Your Commodore 16 interprets the abbreviation exactly as it would read the full name. An example of an abbreviation is as follows:

L then SHIFT O prints L ┌ on your screen

Your computer reads this as if you'd typed out **LOAD**. Abbreviations are useful as a timesaver. There is a complete list of the accepted abbreviations in the BASIC Encyclopedia.

User Defined Functions

An effective way to use your computer's math capability is to create user defined functions. User defined functions are extremely useful in calculations, and easy to implement using your Commodore 16. User defined functions allow you to program a formula, and then let your computer plug in values to be calculated. This can be used for many different purposes.

Here is a statement utilizing the user defined function for calculating the value of a secant:

```
10 DEF FNS(X)=1/COS(X)
```

This figures out the value of the secant of any number entered for X. FNS is the name of the function defined by this statement. Appendix C contains a table of mathematical functions not included in your computer's BASIC language.

User defined functions save memory space when you would use a function more than once, and make your programs easier to read and understand.

CHAPTER 6

GRAPHICS AND COLOUR

- Graphics characters
- Character animation
- Controlling colours
- High resolution graphics
- Points, lines, and labels
- Squares, circles, polygons, and painting
- Multi-colour graphics

GRAPHICS CHARACTERS

You should remember from Chapter 2 that each letter key contains 2 different graphic characters, as do the @, −, *, and £ keys for a total of 62 different graphic characters. To print these graphic characters, you must hold down the SHIFT or C= keys while you press the key for the graphics symbol you want.

When your Commodore 16 is not in typing mode, hold down SHIFT and press a letter key to print the graphics character on the right side of that letter key. These characters include the playing card suits, a solid and a hollow ball, and a set of lines and connecting characters that you can use to draw many different pictures on your screen.

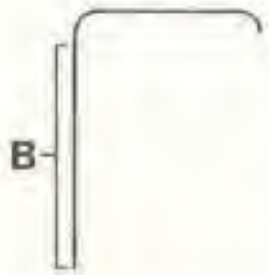
The C= key pressed with a letter key always produces the graphic character pictured on the left front of the key. The left side graphics are an assortment of bars, squares, lines and blocks, useful in drawing charts and graphs.

Pick a Card, Any Card

Here is an example to help give you a better idea of how to use the graphic characters to create representations. Follow these instructions to create a playing card, in this case the six of hearts.

First, change the cursor colour to red. Hold down the CTRL key along with the "3" key to change the cursor colour.

Now it's time to draw the top edge of the card: Press the SHIFT LOCK key so that it stays down. Press "U", followed by the "C" key 5 times, and then the "I".



To draw the left side: Keeping the SHIFT LOCK down, go to the space directly beneath the curved edge (where you pressed the "U" in the last step. Press the "B" key, then cursor to the space directly beneath the line segment you just drew and press "B" again. Do this until you've pressed the "B" key a total of 7 times. (Using the cursor keys will be tough at first; you'll get better with practice.)

To draw the bottom edge: Similar to the top edge, you press "J" once and then press the "C" key 5 times, finishing off with the "K" key.

To draw the right edge: Draw this in the same fashion as the left edge, pressing the 'B' key a total of 7 times (while cursoring to the appropriate place).

Release the **SHIFT LOCK** by pressing it once. In the upper left corner (one space down and one space over from the rounded edge) press the '6'. Go to the bottom right corner, position the cursor one space up and space to the left and type another '6'.

Now to get down to the heart of the matter—6 of them to be exact. Press the **SHIFT LOCK** down again. Go one space down from and one space to the right from where you typed the 6. Press the 'S' key, hit the **SPACE BAR**, and press the 'S' again. Cursor down two rows and repeat the 'S-**SPACE**-S' sequence. Add the last two hearts two more rows down. You now have an official six of hearts playing card. This won't do you too much good in a game of blackjack, however. If you want to improve your hand, create a couple of cards of your own.

NOTE: When the **SHIFT LOCK** is down, you can press **RETURN** and not get a SYNTAX ERROR from your computer as a response. Even though you had characters on the line that weren't commands that the computer can understand, your computer interprets the **RETURN** key as just an instruction to return to the beginning of the next line when the **SHIFT** key is held down. Your computer does not try to read or interpret what has been typed as BASIC language commands when **SHIFT** or **SHIFT LOCK** is down.

You can use the graphic keys to enhance your printout. For example, here's how to underline a word or column:

First, move the cursor to the line below what you want to underline. Then hold down the **C=** key and the "T" key, which prints an underline space. Hold these two keys down until the entire portion of text is underlined.

The purpose of this section is to show you how the graphic symbols of your Commodore 16 can be manipulated to create different shapes and figures and used in a more practical sense. In addition to the 82 graphic characters available to you, you can also use the reverse of these characters. (Remember the racing stripe!) Now that you have a good idea of what is involved in using the graphic symbols to build different forms, you should experiment with them yourself, and see what you come up with.

CHARACTER ANIMATION

Movies are really a sequence of still pictures. Each picture is a little different from the one that came before. The projector shows each picture for a very short time, then goes on to the next one. The scene becomes animated.

Computer animation works the same way. First the computer draws one picture, then it changes the picture slightly. Your Commodore 16 is fast enough to move objects smoothly around the screen in your games and practical programs. A movie is animated at a rate of 30 pictures per second. The changes are fast enough to fool the eye and create the illusion of movement. The only way you can attain the speed to create

this illusion on your computer is to use a program to draw a picture, wait for a split second, then change to a new picture.

To get the program to create pictures you would use the PRINT statement with the graphic characters. The simplest type of animation involves alternating two characters to get the effect of movement.

This program simulates animation by alternating the circle (SHIFT & O) and heart (SHIFT & S) symbols. If you use your imagination, you could consider this a heartbeat of sorts.

IMPORTANT TO NOTE: Each time SHIFT or C= is referred to, it should be typed at the SAME TIME as the key following it when entering the program, since nothing happens when either is typed by itself.

Remember to type **NEW** and press RETURN before entering each new program, and press RETURN to enter each line in all these programs.

```
10 PRINT "HOME SHIFT Q"
20 FOR L=1 TO 100
30 NEXT L
40 PRINT "HOME SHIFT S"
50 FOR M=1 TO 200
60 NEXT M
70 GOTO 10
```

type these keys simultaneously

You'll see the obvious limitation of animation alternating two characters when you RUN this program. To stop it, press the RUN/STOP key.

To get a more interesting effect you can build a small picture from several graphic characters, then change a few of the characters while leaving others in the same place. This gives the effect of part of a larger object moving, as in the following program:


```
10 PRINT " HOME SHIFT M SHIFT W SHIFT N "
20 PRINT " SPACE C= + SPACE "
30 PRINT " SHIFT N SPACE SHIFT M "
40 FOR L=1 TO 300: NEXT L
50 PRINT " HOME SPACE SHIFT W SPACE "
60 PRINT " C= T C= + C= T "
70 PRINT " SPACE C= G C= G "
80 FOR L=1 TO 300: NEXT L
90 GOTO 10
```

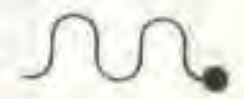
Type **RUN** and press RETURN.

In both examples of animation so far, the figure has been stationary in one area on the screen. The next step is to move the animated figure around. The TAB function helps when you want to move objects from the left edge. (The TAB function is explained in detail in the BASIC Encyclopedia.) The following program portrays a snake crawling on the screen:

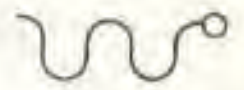
Remember that SHIFT and the following key are still typed together.

```
5 FOR A=0 TO 30
10 PRINT " SHIFT CLR "
20 PRINT TAB (A) " SHIFT U SHIFT I SHIFT U SHIFT I "
30 PRINT TAB (A) " SHIFT K SHIFT J SHIFT K SHIFT I SHIFT Q"
40 FOR L=1 TO 100: NEXT L
50 PRINT " SHIFT CLR "
60 PRINT TAB (A+1) " SHIFT I SHIFT U SHIFT I SHIFT U SHIFT W"
70 PRINT TAB (A+1) " SHIFT J SHIFT K SHIFT J SHIFT K "
80 FOR L=1 TO 100: NEXT L
90 NEXT A
```

Lines 20 and 30 should look like this when typed in:



Lines 60 and 70 will look like this:



Using characters like the ball (SHIFT Q), you can play video games on the screen. To move a ball, just erase the ball and replace it at a new position, as in this program:

```
10 PRINT " SHIFT CLR "
20 PRINT " SPACE SHIFT Q■■";
30 FOR L=1 TO 50: NEXT L
40 GOTO 20
```

Left cursor arrow

When you RUN the program, remember to press the **RUN/STOP** key when you want to stop moving the ball.

CONTROLLING COLOURS

Separate colours can be put into each part of the screen. The border can be one colour, the background a different one, and each character can have its own colour. You already know how to set the character colours using the keyboard. You can change the colours of the other screen areas using the BASIC language command COLOR.

For example, you can turn the border of your screen red by typing the command **COLOR 4,3** and pressing the **RETURN** key. The number 4 in the command stands for the border area, and colour number 3 is red (the same number as on the key marked RED).

Try typing **COLOR 0, 7** and hitting **RETURN**. The screen background now turns blue. The number 0 stands for the background, while the 7 is blue (also the same as the keyboard).

The first number after the word **COLOR** specifies the area on the screen you want to change. This table lists what each screen area number controls; you'll learn about areas 2 and 3 when you get into multi-colour graphics later in this chapter.

Screen Area Numbers

AREA #	AREA NAME
0	Background
1	Character
2	Multi-colour 1
3	Multi-colour 2
4	Border

The second number after **COLOR** selects the colour you want to put on the area of the screen you've specified. The colour numbers correspond to the colour keys on the keyboard.

Colour Numbers

COLOUR #	COLOUR	#	COLOUR
1	BLACK	9	ORANGE
2	WHITE	10	BROWN
3	RED	11	YELLOW GREEN
4	CYAN	12	PINK
5	PURPLE	13	BLUE GREEN
6	GREEN	14	LIGHT BLUE
7	BLUE	15	DARK BLUE
8	YELLOW	16	LIGHT GREEN

Each colour also has an adjustable brightness level, called the *luminance*. You can add a number from 0 (darkest) through 7 (brightest) after the colour number to vary the colour. Type **COLOR 4 ,3, 0** and press **RETURN**. The border becomes a dark red. Type **COLOR 4, 3, 7** and the border changes to a bright red.

In short, the COLOR command looks like this:

COLOR area, colour, luminance

Here is a quick program to show you all your Commodore 16's colours:

First type **NEW** and press **RETURN**.

```
10 COLOR 0, 7, 7
20 FOR M=0 TO 7
30 FOR N=1 TO 2
40 FOR L=1 TO 16
50 PRINT " CTRL RVS ON " ;        type these keys together
60 READ A
70 COLOR 1, A, M
80 PRINT "SPACE SPACE";
90 NEXT L
100 PRINT
110 RESTORE
120 NEXT N,M
130 COLOR 1, 2, 4
200 DATA 7,14,4,13,6,16,11,8,10,9,3,12,5,15,2,1
```

When you RUN this program, the screen background changes to a light blue, and the spectrum of Commodore 16 colours is shown at each luminance level. You'll notice that black is the same at all luminance levels.

NOTE: Like most of the BASIC graphic terms reviewed in this chapter, COLOR may be referred to as a statement or command interchangeably.

THE GRAPHIC COMMAND

The graphics you've seen so far use only the keyboard without really taking advantage of your computer's capabilities. The BASIC language of your Commodore 16 contains commands to draw shapes and forms through programs. To use the graphics-related commands of your computer, you must enter a new mode, the **GRAPHIC** mode. Graphic mode can be considered the drawing mode, since all the drawing commands are "activated". You can't use these commands until you specify exactly what graphics mode you want to use; you specify which type of graphics mode you want by using the **GRAPHIC** command. There are three different modes: normal text, high-resolution graphic and multi-colour graphic modes. With the **GRAPHIC** command, you can even have part-text, part-graphic screens that let you can write on one part of the screen and draw on the rest. The command to enter this new mode is **GRAPHIC**.

In general the **GRAPHIC** command looks like this:

GRAPHIC mode, clear ← this part is optional

Mode number	Effect
0	Text
1	High-resolution graphics
2	High-resolution graphics + text
3	Multi-colour graphics
4	Multi-colour graphics + text

Clear number	Effect
0	Don't clear screen
1	Clear screen

To switch from the normal graphics (of text mode) to high-resolution, just type the command **GRAPHIC 2,1** and press RETURN. The screen goes blank and the cursor reappears near the bottom of the screen. Your Commodore 16 screen is divided into 2 separate sections: the top for graphics and the bottom five lines for text. If you don't want the bottom five lines for text, you can use the command **GRAPHIC 1,1**, but you won't be able to see any commands you type. You can switch back and forth from graphics to text using the **GRAPHIC** command. The command **GRAPHIC 0** switches the screen back to text, while **GRAPHIC 2** switches back to high-res without erasing the screen. Adding ,1 after the command erases the screen.

There is another way to clear the high-resolution screen. The command **SCNCLR** erases the screen without changing the graphic mode. Once you use high-resolution graphics, the computer sets aside 10K of memory for your high-res screen. This memory is taken from the BASIC program area. When you are through using graphics, you can reclaim this memory by using the command **GRAPHIC CLR**.

HIGH RESOLUTION GRAPHICS

Your Commodore 16 screen contains 25 rows of 40 characters each, or 1000 total character positions on the screen. Each character is formed out of single dots, with 8 rows of 8 dots each making an entire character. Your screen has a total of 320 dots on each row, and 200 rows of dots, or 64,000 dots all together. The high resolution graphics on your Commodore 16 give you control over every single dot.

Using normal graphics, you have limited control over the individual dots. Drawing a racing stripe or a playing card is a nice little exercise, but your creation is limited to using the characters (letters, graphic symbols, etc.) on your computer keyboard. You can still create all kinds of shapes and figures, but just a fraction of what you could do if you could control each dot by itself. The high resolution graphics capability of your Commodore 16 lets you do just that. Resolution

refers to the precision and control you have in drawing; with high resolution graphics, you can use commands that let you draw and erase dots, lines, circles, and other shapes.

There is one limit to high-resolution (hi-res for short) graphics. Your computer can only use two colours in each 8 x 8 character position. That is, each 8 x 8 space on the screen (where a single character would fit) is limited to two colours (foreground and background colour for that square). You can use different colours for each different character position, but only two colours within that position. Another graphic mode that will be covered later in this section, multi-colour mode, allows up to four different colours per character position at the cost of the resolution available in the high-resolution mode.

Here is a program that utilizes some of the high resolution graphics capability of your computer and the **GRAPHIC** command in particular. Clear the computer's memory with the **NEW** command so you're sure that there are no "left-over" program lines, and then type:

```
10 COLOR 0,1
20 GRAPHIC 1,1
30 FOR L=2 TO 16
40 COLOR 1,L,2
50 DRAW 1,0,L*12 TO 319,L*12
60 DRAW 1,L*18,0 TO L*18,199
70 NEXT L
80 FOR L=1 TO 9000:NEXT
90 COLOR 1,2,7
100 GRAPHIC 0
```

Notice that the colours change near the intersections. This is due to the limitations of hi-res graphics, with too many colours too close together.

Points, Lines and Labels

Type the commands **GRAPHIC 2,1**: **DRAW 1,0,0** and press **RETURN**. Look closely at the upper left corner of the screen. Your Commodore 16 drew a white dot there. The DRAW command can be used to draw a single dot anywhere on the screen, a line, or a shape.

Some forms of the **DRAW** command are:

COMMAND	RESULT
DRAW colour source, column, row	POINT
DRAW colour source, column, row **TO** column, row	LINE
DRAW colour source **TO** column, row	LINE DRAWN FROM LAST POINT

Colour source is 0 for the background, 1 for the foreground. Anything drawn in the background colour (0) is the same as erasing the foreground colour at that spot.

In the DRAW command, the first number is either 1 (draw a dot) or 0 (erase a dot). The next two numbers are for the column and row positions for the dot. So if you wanted to draw a dot at column 17, row 20, you would type **DRAW 1,17,20**. To erase the same dot, you'd type **DRAW 0,17,20**.

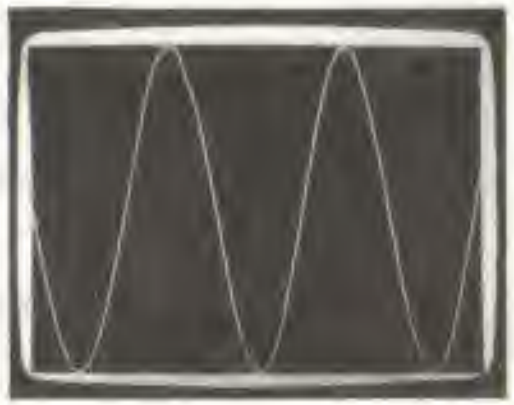
The DRAW command can also draw a line between any two points. Just add the word TO and the coordinates of the other end, like this: DRAW 1,1,1 TO 100,100. This draws a line from the dot at 1,1 to 100,100. You can erase this line by typing the same command, just substituting a 0 for the 1 immediately after DRAW.

If you are used to drawing graphs in math, you might get a little confused at first while using the computer. The coordinate system in your Commodore 16 is different from what you've used before. In math the 0,0 point would either be at the centre or the lower left corner of the screen, but on your computer it is the **upper** left corner. You'll get used to this system as you practice.

Once you have put a point or line on the screen, you can draw a line from it to any other point like this: **DRAW 1 TO 150,50**. This draws a line from the last point drawn to column 50 row 150. If your program uses a lot of **DRAW TO** commands, you could place the first dot at a position on the screen by using the **LOCATE** command, as in **LOCATE 100,100**, to immediately return to the first position.

This program draws a curve based on the sine function.

```
NEW
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 LOCATE 0,100
50 FOR X=1 TO 319
60 Y = INT (100+99 * SIN(X/20))
70 DRAW 1 TO X,Y
80 NEXT X
90 FOR L=1 TO 5000
100 NEXT L
110 GRAPHIC 0
```



Don't type NEW after RUNning the last program. To plot the program differently,change line 70 to:

```
70 DRAW 1, X, Y
```

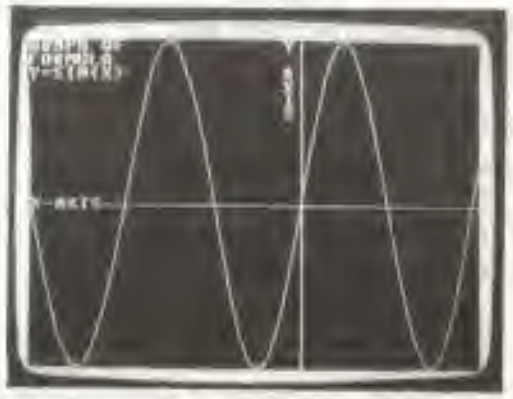
This program plots the same curve using points instead of lines.

The Char Command

Graphs are more easily understood and useful if you label them. You can use the CHAR command to mix text right into a high resolution drawing. For instance, the statement **CHAR 1,0,5,"HELLO"** puts the word HELLO into the sixth row at the left edge of the screen (the first row is '0'). The first number after the word CHAR is either 1 (for draw) or 0 (for erase). The next two numbers are the column and row where the text appears.

Leave the last two programs in the computer: don't type NEW. Add these lines:


```
81 CHAR 1,0,0,"GRAPH OF":CHAR 1,0,1,"FORMULA"
82 CHAR 1,0,2,"Y=SIN(X)"
83 DRAW 1,0,100 TO 319,100,189,0 TO 189,199
84 CHAR 1,0,12,"X-AXIS": CHAR 1,22,0,"Y"
85 CHAR 1,22,2,"A": CHAR 1,22,3,"X"
86 CHAR 1,22,4,"I": CHAR 1,22,5,"S"
```



SQUARES, CIRCLES, POLYGONS AND PAINTING

Using the DRAW command, you can draw pictures by plotting many dots or lines. To draw a square, you can use the command DRAW 1,0,0 TO 100,0 TO 100,100 TO 0,100 TO 0,0 to connect four dots with four lines (plotting each endpoint of the square) or you can just use the **BOX** command.

Drawing Rectangles

One of the graphic commands of your Commodore 16 makes it easier to draw squares and other rectangular shapes. The **BOX** command lets you pick the points of 2 opposite corners of the square. To duplicate the same box as in the above example, just use **BOX** 1,0,0,100,100. The number 1 again means that you want to draw and not erase. The next four numbers are the coordinates of the box's opposite corners, (0,0) at the upper-left corner and (100,100) near the middle of the screen.

The **BOX** command can form any rectangle just by changing the corners. You can even rotate the box by specifying an angle (in degrees) after the last coordinate, like this: **BOX** 1,50,50,100,100,45. The box is rotates 45 degrees clockwise so that it resembles a diamond shape.

If you would like to draw a solid box instead of just the outline, you just add a comma 1 after the angle. A solid box at the centre of the screen is shown as **BOX** 1,100,50,220,150,,1. Notice that you need a comma

to take the place of the angle, even though you don't want the box rotated. This is so your computer reads the comma as the DEFAULT value, which means that it interprets your lack of response as an instruction. If you don't include the comma, the 1 at the end of the line is read as the angle for the box to be rotated.

Some typical forms of the **BOX** command are:

Command	Effect
BOX on, column 1, row 1, column 2, row 2	Outline
BOX on, col 1, row 1, col 2, row 2, angle	Rotated
BOX on, col 1, row 1, col 2, row 2, , fill	Solid box
BOX off, col 1, row 1, col 2, row 2, angle, fill	Erase area of screen

Column 1, row 1, etc. are screen positions of endpoints that you specify. Column 1, row 1 is the upper left corner of the box, while row and column 2 is the bottom right corner.

Here is a program that illustrates the **BOX** command (line 60):

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 2,1
40 A = RND(1)* 20+ 10
50 FOR L=0 TO 359 STEP A
60 BOX 1, 100, 30, 220, 130, L
70 NEXT L
80 FOR L=1 TO 2000: NEXT L
90 GRAPHIC 0,1
```

Drawing Circles

Your Commodore 16 also has commands for drawing circles. Like the **BOX** command, you can vary the shape of the circle (to form an oval or an ellipse), and you can rotate the oval. You can also just draw a section of the shape (called an arc).

The usual forms of the **CIRCLE** command are:

Command	Effect
CIRCLE on, centre column, centre row, radius	circle
CIRCLE on, c-col, c-row, width, height	oval
CIRCLE on, c-col, c-row, wid, ht, start, finish	arc
CIRCLE on, c-col, c-row, width, height,,,angle	rotated oval
CIRCLE on, c-col, c-row, wid, ht,,,,point angle	polygon

This command draws a circle in the centre of the screen: **CIRCLE 1,160,100,50.** This tells your computer to draw a circle with its centre at column 160 and row 100, with a radius of 50. This may actually produce an oval, since the dots on some TV's and monitors (American ones, for example) are taller than they are wide. To change this to a real circle you must add a separate number to tell that the height is different from the width, like this: **CIRCLE 1,160,100,50,42.**

Drawing Polygons

Your Commodore 16 can also draw a square, triangle or other polygon using the **CIRCLE** command. Just tell the computer how many degrees to go between points on the circle, like this: **CIRCLE 1,160,100,50,42,,,120.** This command draws a triangle, since each side is 120 degrees. (Remember that omitting number values while including commas in a graphic command causes your computer to read standard default values for the missing number.) A simple formula to get the angle for a polygon with N sides is 360/N.

Here's a quick program for drawing polygons:

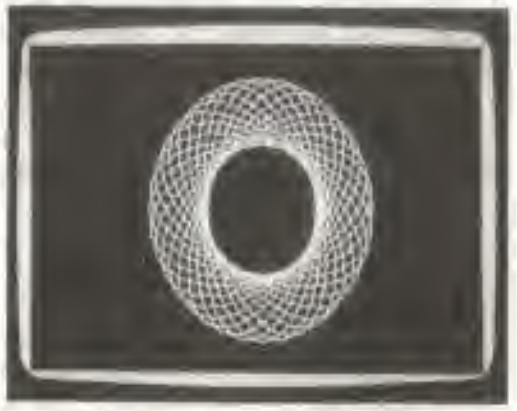
```
10 GRAPHIC 2,1
20 INPUT"HOW MANY SIDES";A
30 IF A<2 OR A>100 THEN PRINT "DON'T BE RIDICULOUS":
   GOTO 20
40 CIRCLE 1,160,80,40,33,,,,360/A
50 GOTO 20
```

You can choose to draw only an arc instead of a whole circle. The **CIRCLE** command accepts the starting and ending angles in degrees,

right after the height number. The command **CIRCLE 1,160,100,50,42,90,180** displays only the lower right section of the circle. To rotate an oval, add the angle of clockwise rotation after the command, like this example: **CIRCLE 1,160,100,100,20,,,30.**

Here's a program that uses **CIRCLE** commands for an interesting effect. Don't forget to clear your computer's memory by typing in **NEW** if you entered the last program.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 A = RND(1)* 20+ 10
50 FOR L=0 TO 359 STEP A
60 CIRCLE 1, 160, 100, 80, 40,,,L
70 NEXT L
80 FOR L=1 TO 2000: NEXT L
90 GRAPHIC 0,1
```



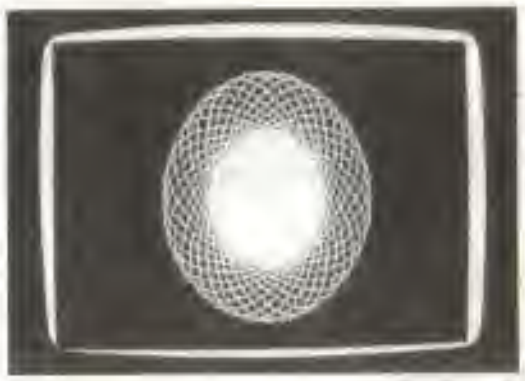
This program will be slightly different each time you **RUN** it.

The Paint Command

To make a circle or other shape more than just an outline, you can use the PAINT command. The PAINT command fills in any enclosed area up to the boundaries formed by lines drawn on the screen. If there are no lines drawn, the screen is filled right to the edge. The BOX command has a fill option that colours in boxes and rectangles. The PAINT command can colour in irregular shapes and other non-uniform areas on the screen that can't be filled with other commands.

To see what the PAINT command does, add this line to the last program:

75 **PAINT 1, 160, 100**



MULTI-COLOUR GRAPHICS

The Commodore 16 high resolution graphics give you control over every single dot or "pixel" on the screen, but you have seen that the ability to put colours close together is limited. Most hi-res programs can use only one or two colours. For including more different colours, your computer has a special "in-between" graphics mode called multi-colour graphics. In multi-colour graphics, you control half as many dots on each row as in hi-res because each dot is twice as wide. You get 160 dots on each row, while still getting 200 rows. There is a trade-off for the use of multiple colours, which is slightly lower resolution.

To begin using multi-colour graphics, review the **GRAPHIC** command earlier in this chapter. You'll see that the multi-colour screen without text is **GRAPHIC 3** and the multi-colour screen with 5 lines of text is **GRAPHIC 4**.

Now look at the table listing the **COLOR** command. There are two areas that we haven't used yet, areas 2 and 3. These areas hold two extra colours. You can use any of the three colours (1, the text colour; 2, an extra colour; and 3, another extra colour). These colours do not interfere with each other on the screen the way the hi-res colours do in some previous programs in this chapter.

This program makes use of multi-colour graphics, showing a "neon sign" effect.

```
NEW
10 COLOR 0,1
20 GRAPHIC 3,1
30 COLOR 3,1
40 TRAP 200
50 DRAW 3,10,10 TO 10,100: DRAW 3,10,55 TO 30,55
60 DRAW 3,30,10 TO 30,100: DRAW 3,50,10 TO 80,10
70 DRAW 3,65,10 TO 65,100: DRAW 3,50,100 TO 80,100
80 FOR L=0 TO 7
90 COLOR 3,2,L
100 FOR M=1 TO 100: NEXT M
110 NEXT L
120 COLOR 3,1
130 FOR M=1 TO 100: NEXT M
140 GOTO 80
200 GRAPHIC 0: COLOR 1,2,7
```

Colour area 3, the second of the multicolour areas, has a special ability that none of the others has. Once you have drawn on the screen using area 3, you can change that colour *everywhere* it appears on the screen by using the COLOR command. If you set the colour with **COLOR 3,5** and draw using that colour, your graphics appear in purple. If you then change the colour with **COLOR 3,6,** all the purple areas would change to green. This doesn't work with any other area. The Series 264 Programmer's Reference Guide contains more information about graphics.

CHAPTER 7

SOUND AND MUSIC

- Introduction
- Volume command
- Sound command
- Creating sound effects
- Making some music
- The Music Machine

INTRODUCTION

Here is a short program to make music on your Commodore 16. After you've typed it in, when you RUN it, a question mark will appear on your screen. Type any number from 0 to 1015 as your response and press **RETURN**. To stop the program, enter a zero as your value.

```
10 VOL 8
20 DO
30 INPUT X
35 IF X>1015 OR X<0 THEN PRINT "0 TO 1015, PLEASE": GOTO 30
40 SOUND 1, X, 10
50 LOOP UNTIL X=0
```

Pressing the **RUN/STOP** key also stops the program.

Here's how to play a single note on your Commodore 16.

First: Type **VOL 8** and press **RETURN**.

Second: Type **SOUND 1,266,60** and press **RETURN**.

You should hear a note play for about a second and then stop. You might consider this an unfinished symphony in the most extreme sense of the word (one note down, 3,500 to go). If you don't hear anything, turn up the volume of your television or monitor and try it again.

These two steps are the only commands that you need to know to play music on your Commodore 16. Both commands are easy to understand and easier to use.

The Volume Command

The **VOL** command controls the VOLume of the notes that your Commodore 16 plays. The number that comes after **VOL** is the setting for the volume. This command works pretty much like the volume knob on your TV. When it is set at zero (**VOL 0**), the volume is off and you won't hear anything. When you set it at 8 (**VOL 8**), the volume is turned up all the way, and your computer plays as loud as it can.

Try the first example again and use a different number after the **VOL** command. The larger the number, the louder the note is played.

The Sound Command

The **SOUND** command tells your computer everything it needs to know about the sound you want to play. The **SOUND** command is followed by three numbers that describe the note.

SOUND voice, note value, duration

The first number in the sound command refers to voice. The number for voice can be a 1,2 or 3. The Commodore 16 sound is produced by two different "voices", 1 for the first voice and 2 for the second voice. The third voice option applies to voice 2's capacity to produce either a tone or noise.

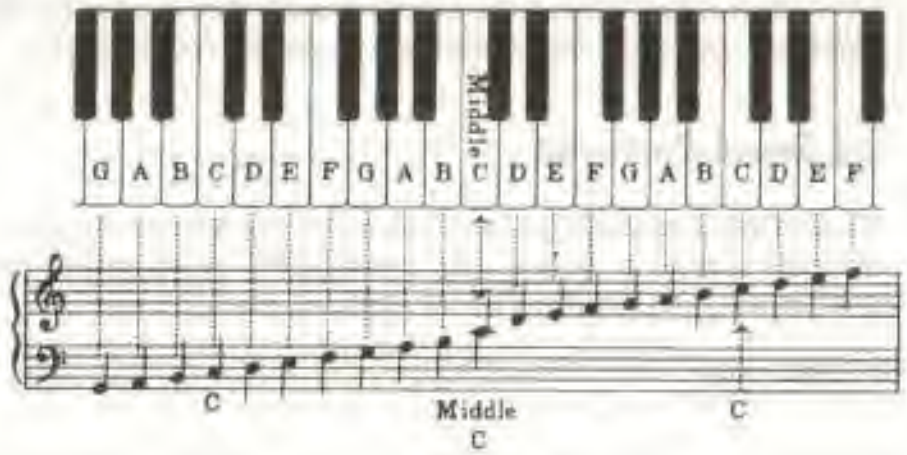
Voice 1 - This voice plays only tones. Select this voice with a 1 after the **SOUND** command.

Voice 2 - This voice is like voice 1, but can be used to play tones or noise for sounds. Type a 2 in the command to use this voice for tones, or a 3 to use this voice for noise, to make sound effects like thunder and rain.

The second number after the word SOUND is the note value (frequency). This can be any number from 0 to 1015. It tells your Commodore 16 how low- or high-pitched a note to play. As the numbers get larger, the notes get higher. The highest values (in the 1015 neighborhood) are not audible to the human ear.

Note: With voice 3, noise is "white" only in the range of 600-940. You can use register values outside this range to create interesting sound effects.

This displays all of the notes in one scale, along with the note value to use. There is a complete chart of notes for the Commodore 16 in the appendix.



NOTE	A	B	C	D	E	F	G
VALUE	770	798	810	834	854	864	881
ACTUAL FREQUENCY (HZ)	440.4	494.8	522.7	588.7	658	699	782.2

Try the following program:

```
NEW
10 VOL 8      <-- sets VOLume
20 X = 0
30 DO
40 SOUND 1,X,5      <-- plays note
50 X = X + 5
60 LOOP UNTIL X = 1015
70 VOL 0      <-- turns off VOLume
80 END
```

This program shows off some of your Commodore 16's musical range.

The third number after the word **SOUND** controls the duration (length) of the note. This tells your computer how long to play the note. This number can be anything from 0 to 65535. This number sets a timer,

which counts time in sixtieths of a second. A duration of 60 keeps the note on for one second. The rule of thumb for duration is the larger the number, the longer the note stays on. In fact, if you use 65535, the note stays on for over 16 minutes. To turn a sound off, use a zero duration, which does not allow the sound to be produced.

A Musical Sound Effect

Sound effects can be created using either musical tones or noise. Combining simple BASIC programs and sound commands can generate unusual and entertaining effects. For instance, the FOR . . . NEXT . . . STEP loop can be used creatively in sound effects. This command sets up a loop; each time the computer reads FOR, it changes the counter variable (S in this example). When it reads NEXT, it goes back to the FOR statement. This program uses a FOR . . . NEXT loop with a negative STEP, to count down from a high number to a lower one by 25 at a time.

```
10 VOL 8      <-- sets VOLume at 8
20 FOR S=1000 TO 700 STEP -25      <-- creates loop, with
30 SOUND 1, S, 1                       downward STEPs
40 NEXT S
```

Type **RUN** and press **RETURN** to hear the sound effect. The key is line 20, which selects a number range from 1000 to 700 going down the scale, STEPping down 25 numbers at a time. Finally, line 30 instructs your Commodore 16 to play each note for just an instant by setting the DURATION to 1, which is 1/60 of a second. Experimenting with different number and duration values can give you some very interesting effects.

Creating a Noise Sound Effect

Using a value of 3 when selecting a voice in the **SOUND** command specifies noise. This is used to create sound effects with noise rather than tone. The following program uses voice 3 to create the sounds of a windstorm.

```
10 VOL 2      <-- sets VOLume level
20 R=INT(RND(0)*10)+1      <-- selects RaNDom number from 1
30 FOR X=1 TO R                to 10
40 SOUND 3, 600+30*X, 10
```

(continued on next page)

```
50 NEXT X
60 FOR X=R TO 1 STEP -1
70 SOUND 3, 600+30*X, 10
80 NEXT X
90 T=INT(RND(0)*100)+30
100 SOUND 3, 600, T
110 GOTO 20
```

Lines 30 and 60 set up FOR . . . NEXT loops for the note value (frequency) of the sound, one increasing and one decreasing, based on the random number from line 20. It is important to have variation in pitch, since windstorms have different forces of gusts of wind. Lines 40 and 70 are the **SOUND** commands that create the noise. Lines 90 and 100 set up a random delay to recreate the uneven nature of a windstorm with time lapses between howls. The program selects a RaNDom number that is used for the duration of another **SOUND** command. This **SOUND** command stays at the same pitch and provides a consistent background noise that serves as a counterpoint to the gusts of wind. This may seem pretty complicated, like you need to be an experienced programmer to be able to do it. But it's really nothing more than experimenting, trying different things, and seeing what noise comes out. Creating sound effects using noise is challenging, trying to capture the right elements of the sound you want exactly. To be good at it, you have to be willing to experiment.

Making Some Music

You may not understand everything that's going on in these programs, but type them in anyway and see what happens.

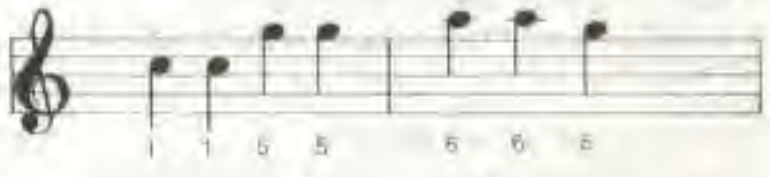
This program simulates a piano using the keys from 1 through 8:

```
5 SCNCLR
10 FOR X=1 TO 8: READ N(X): NEXT X
20 VOL 8
30 DO
40 GET A$: IF A$="" THEN 40
50 A=ASC(A$): IF A<49 OR A>56 THEN 90
60 N=A-48
70 SOUND 1, N(N), 5
80 COLOR 4, N, 3
90 LOOP UNTIL A=32
100 VOL 0: COLOR 4, 2, 7
110 DATA 169, 262, 345, 383, 453, 516, 571, 596
```


Press numbers 1 through 8 to play notes. The screen border even changes colours with the different notes. When you finish playing, press the space bar to stop the program.

Now that your Commodore 16 can be used like a piano, you might feel the urge to play a popular song. Here are the numbers to press to play a well-known song that surely must be considered a classic.

```
1 1 5 5 6 6 5
4 4 3 3 2 2 1
5 5 4 4 3 3 2
5 5 4 4 3 3 2
1 1 5 5 6 6 5
4 4 3 3 2 2 1
```



This next program plays a song by reading a list of **DATA** statements. Your computer **READ**s the numbers contained in the **DATA** statement as values for variables (in this case X and Y), changing the values with each loop. The **DATA** statements are in pairs. The first number is the note value for the **SOUND** command and the second number is the duration for the **SOUND** command.

Row Boat

```
10 VOL 8
20 DO
30 READ X, Y
40 SOUND 1, X, Y
45 FOR D=1 TO Y*16+30:NEXT
50 LOOP UNTIL X=0          <-- This loop creates a brief delay between notes
60 END
100 DATA 169, 45, 169, 45, 169, 30
110 DATA 262, 15, 345, 45, 345, 30
120 DATA 262, 15, 345, 30, 383, 15
130 DATA 453, 60, 596, 45, 453, 45
140 DATA 345, 45, 169, 45, 453, 30
150 DATA 383, 15, 345, 30, 262, 15
160 DATA 169, 60
200 DATA 0, 0
```

THE GREAT COMMODORE 16 MUSIC MACHINE

The last program is a little longer. This is the "GREAT COMMODORE 16 MUSIC MACHINE". When you press a key from 1 through 9, the note is played, and a note appears on the staff on the correct line.

```
5 GOSUB 1000
6 FOR X=1 TO 9: READ N(X): NEXT X
8 CHAR 1, 8, 1,"*THE GREAT MUSIC MACHINE*"
10 VOL 7
20 DO
30 GET A$: IF A$="" THEN 30
35 A=ASC(A$): IF A<49 OR A>57 THEN 50
36 N= A - 48
40 SOUND 1, N(N), 4
45 GSHAPE N$, 150, 8 * (6+(9-N)), 4
46 FOR Z=1 TO 50: NEXT Z
47 GSHAPE N$, 150, 8 * (6+(9-N)), 4
50 LOOP UNTIL A=32
55 VOL 0: GRAPHIC 0: SCNCLR
60 END
100 DATA 345, 383, 453, 516, 571, 596, 643, 685, 704
1000 GRAPHIC 1,1
1010 FOR Y=60 TO 124 STEP 16
1020 DRAW 1, 100, Y TO 200, Y
1030 NEXT Y
1040 A$="FEDCBAGFE"
1050 FOR X=1 TO 9: C=13
1060 IF INT(X/2)= X/2 THEN C=14
1070 CHAR 1, C, X+6, MID$ (A$, X, 1), 0
1075 CHAR 1, C+10, X+6, RIGHT$ (STR$ (10-X), 1)
1080 NEXT X
1090 FOR X=1 TO 8: FOR Y=11 TO 16: DRAW 1, X, Y: NEXT Y, X
1100 Y=1: X=8: DRAW 1, 8, 16 TO X, Y
1110 SSHAPE N$, 1, 1, 8, 16
1120 GSHAPE N$, 1, 1, 4
1130 RETURN
```

no space

As you can see, music and sound can be used to enhance programs or be the focus of the program itself. The examples in this chapter just give you a taste of the music capabilities of your Commodore 16. Don't be afraid to try new sounds and noises and create your own masterpiece.

By now, you've begun to get a handle on some of the things you can do with your Commodore 16. The aim of this manual has been to give you a taste of computing, to have you try new things and see what happens, to learn about your computer and to have fun in the process. But there's still so much more . . . programming, for example. This manual has touched on how to begin to use BASIC, but is not a tutorial or BASIC textbook by any means. The BASIC Encyclopedia (immediately following this chapter) gives a complete listing of ALL the BASIC commands on your Commodore 16, with explanations and examples. In many of the programs in the last couple of chapters, you may not have understood exactly what steps were doing what. To learn more about programming with the BASIC language, you should read books that were written to teach BASIC. A list of these books appears in the appendix. If you are really interested in programming, you might want to get the Series 264 Programmer's Reference Guide, which focuses on the hows and whys of programming, revealing the secret and not-so-secret tricks of the programming trade. Now that you've finished reading this manual, you might think that you're on your own, but not entirely. You can read any of the many computer magazines, some of which are entirely devoted to Commodore computers, peripherals and software. You can also become part of a group of people near you who share your interest in Commodore computers: a Commodore User's Group. For more information on these groups, contact the CBM Information Centre.

BASIC 3.5

ENCYCLOPEDIA

- Introduction
- Commands
- Statements
- Functions
- Variables and operators
- Abbreviation and reference chart

INTRODUCTION

You've seen in this manual an assortment of exercises using the BASIC language, to give you a feel for computer programming and some of the vocabulary involved. This encyclopedia gives a complete list of the rules and terms (SYNTAX) of the BASIC 3.5 language, along with a concise description of each. Experiment with these commands, and remember that you can't damage your Commodore 16 by typing in programs, and that the best way to learn computing is by doing.

The encyclopedia provides formats and brief explanations and examples of the BASIC 3.5 commands and statements. It is not intended to teach BASIC. If you are interested in learning BASIC, Appendix G lists tutorial books that will help.

Commands and statements are listed in separate sections. Within the sections, the commands and statements are listed in alphabetical order. Commands are used mainly in direct mode, while statements are most often used in programs. In most cases, commands can be used as statements in a program if you prefix them with a line number. You can use many statements as commands by using them in direct mode (i.e. without line numbers). If you are unsure where a term is located, they are all listed in the reference chart.

The BASIC Encyclopedia is organized along the lines of the following:

- **COMMANDS:** the commands used to work with programs, edit, store, and erase them.
- **STATEMENTS:** the BASIC program statements used in numbered lines of programs.
- **FUNCTIONS:** the string, numeric, and print functions.
- **VARIABLES AND OPERATORS:** the different types of variables, legal variable names, and arithmetic and logical operators.

A fuller explanation of BASIC 3.5 commands is provided in the Series 264 Programmer's Reference Guide, available from your Commodore dealer or your local bookstore.

COMMAND AND STATEMENT FORMAT

The commands and statements presented in this section of the encyclopedia follow consistent format conventions to make them as clear as possible. In most cases, there are several examples to illustrate what the actual command looks like. The following example shows some of the format conventions that are used in the BASIC commands and statements:

EXAMPLE: **LOAD**"*program name*",**D***0*, **U***8*

keywords — *argument* — additional arguments (possibly optional)

The parts of the command or statement that you must type in exactly as they appear are highlighted in a darker type in the format listing, while the name of the command is in capital letters. The words that you don't type in exactly, such as the name of a program, are printed in lighter italic type. When quote marks (" ") appear (usually around a program or file name), you should include them in the command or statement, like in the format example.

- KEYWORDS appear in uppercase letters and boldface type. YOU MUST ENTER THESE KEYWORDS EXACTLY AS THEY APPEAR. However, many keywords have abbreviations that you can also use (see the reference chart).

Keywords are words that are part of the BASIC language that your computer knows. Keywords are the central part of a command or statement. They tell the computer what kind of action you want it to take. These words cannot be used as variable names.

- ARGUMENTS (also called parameters) appear in lowercase letters. Arguments are the parts of a command or statement that you select; they complement keywords by providing specific information about the command or statement. For example, a keyword tells the computer to load a program, while an argument tells the computer which specific program to load and a second argument specifies which drive the disk containing the program is in. Arguments include filenames, variables, line numbers, etc.

- SQUARE BRACKETS [] show OPTIONAL arguments. You select any or none of the arguments listed, depending on your requirements.

- ANGLE BRACKETS < > indicate that you MUST choose one of the arguments listed.

- VERTICAL BAR | separates items in a list of arguments when your choices are limited to those arguments listed, and you can't use any other arguments. When the vertical bar appears in a list enclosed in SQUARE BRACKETS, your choices are limited to the items in the list, but you still have the option not to use any arguments.

• ELLIPSIS . . . , a sequence of three dots, means that an option or argument can be repeated more than once.

• QUOTATION MARKS " " enclose character strings, filenames, and other expressions. When arguments are enclosed in quotation marks in a format, you must include the quotation marks in your command or statement. Quotation marks are required parts of a command or statement.

• PARENTHESES () When arguments are enclosed in parentheses in a format, you must include the parentheses in your command or statement. Parentheses are also required when they appear in a command or statement description.

• VARIABLE refers to any valid BASIC variable name, such as X, A$, or T%

• EXPRESSION means any valid BASIC expression, such as A+B+2 or .5*(X+3)

BASIC COMMANDS

AUTO

AUTO [line#]

Turns on the automatic line numbering feature which eases the job of entering programs by typing the line numbers for you. As you enter each program line and press **RETURN** the next line number is printed on the screen, with the cursor in position to begin typing that line. The [line#] argument refers to the increment between line numbers. AUTO with NO ARGUMENT turns off auto line numbering, as does RUN. This statement is executable only in direct mode.

EXAMPLES:

AUTO 10	automatically numbers line in increments of ten
AUTO 50	automatically numbers line in increments of fifty
AUTO	turns OFF automatic line numbering

BACKUP

BACKUP Ddrive# **TO** Ddrive# [, **ON** Uunit#]

This command copies all the files on a diskette to another diskette on a dual drive system. You can copy onto a new diskette without first using the HEADER command to format the new diskette because the BACKUP command copies all the information on the diskette, including the format. You should always BACKUP important diskettes in case the original is lost or damaged.

Because the BACKUP command also HEADERS diskettes, it destroys any information on the diskette onto which you're copying information. So if you're backing up onto a previously used diskette, make sure it contains no programs you wish to keep. See also the COPY command.

NOTE: This command can only be used with dual disk drive.

EXAMPLES:

BACKUP D0 TO D1	Copies all files from the disk in drive 0 to the disk in drive 1
BACKUP D0 TO D1, ON U9	Copies all files from drive 0 to drive 1 in disk drive unit 9

COLLECT

COLLECT [Ddrive#] [,**ON** Uunit#]

Use this command to free up space allocated to improperly closed files and deletes references to these files from the directory.

EXAMPLE:

COLLECT D0

CONT

CONT (Continue)

This command is used to re-start the execution of a program that has been stopped by either using the STOP key, a STOP statement, or an END statement within the program. The program will resume execution where it left off. CONT will not work if you have changed or added lines of the program (or even just moved the cursor to a program line and hit **RETURN** without changing anything), if the program stopped due to an error, or if you caused an error before trying to re-start the program. The error message in this case is CAN'T CONTINUE ERROR.

COPY

COPY [Ddrive#,] "source file" TO [Ddrive#,] "other file" [,ON Uunit#]

COPYs a file on the disk in one drive (the source file) to the disk in the other on a dual disk drive only, or creates a copy of a file on the same drive (with a different file name).

EXAMPLES:

COPY D0,"NOON" TO D1,"NIGHT"	Copies NOON from drive 0 to drive 1, renaming it NIGHT
COPY D0, "STUFF" TO D1, "STUFF"	Copies STUFF from drive 0 to drive 1
COPY D0 TO D1	Copies all files from drive 0 to drive 1

COPY "CATS" TO "DOGS"	Copies CATS as a program called DOGS on the same drive.

DELETE

DELETE [first line#] [- last line#]

Deletes lines of BASIC text. This command can be executed only in direct mode.

EXAMPLES:

DELETE 75	Deletes line 75.
DELETE 10 - 50	Deletes lines 10 through 50 inclusive.
DELETE - 50	Deletes all lines from the beginning of the program up to and including line 50.
DELETE 75-	Deletes all lines from 75 on to the end of the program.

DIRECTORY

DIRECTORY [Ddrive#] [,Uunit#] [,"filename"]

Displays a disk directory on the Commodore 16 screen. Use **CTRL**-S to pause the display (any other key restarts the display after a pause). Use the **C=** key (the Commodore key) to slow it down. The DIRECTORY command cannot be used to print a hard copy. You must load the disk directory (destroying the program currently in memory) to do that.

EXAMPLES:

DIRECTORY	List all files on the disk.
DIRECTORY D1, U9, "WORK"	Lists the file on disk drive unit 9 (8 is the default), drive 1, named WORK.
DIRECTORY "AB*"	Lists all files starting with the letters "AB", like ABOVE, ABOARD, etc.
DIRECTORY D0, "FILE ?.BAK"	The ? is a wild-card that matches any single character in that position: FILE 1.BAK, FILE 2.BAK, FILE 3.BAK all match the string.

NOTE: To print out the DIRECTORY of drive 0, unit 8, use the following.

```
LOAD"$0",8
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

DLOAD

DLOAD "filename" [,Ddrive#] [,Uunit#]

This command loads a program from disk into current memory. (Use LOAD to load programs on tape.) You must supply a program name.

EXAMPLES:

DLOAD "DTRUCK"	Searches the disk for the program "DTRUCK" and LOADs it.
DLOAD (A$)	LOADs a program from disk whose name is in the variable A$. You will get an error if A$ is empty.

The DLOAD command can be used within a BASIC program to find and RUN another program on disk. This is called chaining.

DSAVE

DSAVE "filename" [,Ddrive#] [,Uunit#]

This command stores a program on disk. (Use SAVE to store programs on tape.) You must supply a program name.

EXAMPLES:

DSAVE "DDAY"	SAVEs the program "DDAY" to disk.
DSAVE (A$)	SAVEs to disk program whose name is in the variable A$.
DSAVE "PROG 3",D0,U9	SAVEs the program "PROG 3" to the disk drive with a unit number of 9.

HEADER

HEADER "diskname" ,Ddrive# [,I.d #] [,**ON** Uunit#]

Before you can use a new diskette for the first time you must format it with the HEADER command. If you want to erase an entire diskette for re-use you can use the HEADER command. This command divides the disk into sections called blocks, and it creates a table of contents, called a directory or catalog, on the disk. The diskname can be any name up to 16 characters long. The i.d. number is any 2 characters. Give each disk a unique i.d. number. Be careful when you HEADER a disk because the HEADER command erases all stored data. Giving no i.d. number allows you to perform a quick header. The old i.d. number is used. You can only use the quick header method if the disk was previously formatted, since the quick header only clears out the directory rather than formatting the disk.

EXAMPLES:

HEADER "MYDISK", I23, D0

HEADER "THEBALL", I45, D1, U8

HELP

HELP

The HELP command is used after you get an error in your program. When you type HELP, the line where the error occured is listed, with the portion containing the error displayed in flashing characters.

KEY

KEY [key #, string]

There are eight (8) function keys available to the user on your Commodore 16 computer: four unshifted and four shifted. Your Commodore 16 allows you to define what each key does when pressed. KEY without any parameter specified gives a listing displaying all the current KEY assignments. The data you assign to a key is typed out when that function key is pressed. The maximum length for all the definitions together is 128 characters. Entire commands or a series of commands) can be assigned to a key. For example:

KEY 7, "GRAPHIC0" + CHR$(13) + "LIST" + CHR$(13)

causes the computer to select text mode and list your program whenever the 'F7' key is depressed (in direct mode). The CHR$(13) is the ASCII character for **RETURN**. Use CHR$(34) to incorporate a double quote into a KEY string.

The keys may be redefined in a program. For example:

10 KEY 2,"TESTING" + CHR$(34):KEY3, "NO"

10 FOR I = 1 TO 8: KEY I,
CHR$(I+132):NEXT

defines the function keys as they are defined on the Commodore 64 and VIC 20

To restore all function keys to their default values, reset your Commodore 16 by turning it off and on, or press the RESET button.

LIST

LIST [first line] [- [last line]]

The LIST command lets you look at lines of a BASIC program that have been typed or LOADed into the Commodore 16's memory. When LIST is used alone (without any numbers following it), you get a complete LISTing of the program on your screen, which may be slowed down by holding down the **C=** key, paused by **CTRL**-S (unpaused by pressing any other key), or STOPped by pressing the **RUN/STOP** key. If you follow the word LIST with a line number, your Commodore 16 only shows that line number. If you type LIST with two numbers separated by a dash, the Commodore 16 shows all lines from the first to the second line number. If you type LIST followed by a number and just a dash, it shows all the lines from that number to the end of the program. And if you type LIST, a dash, and then a number, you get all the lines from the beginning of the program to that line number. Using these variations, you can examine any portion of a program, or easily bring lines to the screen for modification.

EXAMPLES:

LIST	Shows entire program.
LIST 100-	Shows from line 100 until the end of the program.

LIST 10	Shows only line 10.
LIST-100	Shows lines from the beginning until line 100.
LIST 10-200	Shows lines from 10 to 200, inclusive.

LOAD

LOAD ["filename" [,device#] [,relocate flag]]

This is the command to use when you want to use a program stored on cassette tape or on disk. If you type just LOAD and hit the RETURN key, the Commodore 16 screen goes blank. Press play, and the Commodore 16 starts looking for a program on the tape. When it finds one, the Commodore 16 prints FOUND "filename". You can hit the C= key to LOAD; if you don't press the key, the computer resumes searching on the tape after a brief interval. Once the program is LOADed, you can RUN, LIST, or change it.

You can also type the word LOAD followed by a program name, which is most often a name in quotes("program name"). The name may be followed by a comma (outside of the quotes) and a number (or numeric variable), which acts as a device number to determine where the program is stored (disk or tape). If there is no number given, your computer assumes device number 1, which is the cassette tape recorder.

The other device commonly used with the LOAD command is usually the disk drive, which is device number 8.

EXAMPLES:

LOAD	Reads in the next program on tape.
LOAD "BASES"	Searches tape for a program called BASES, and LOADS it if it is found.
LOAD A$	Looks for a program whose name is in the variable called A$.

LOAD "BRIDGES",8	Looks for the program called BRIDGES on the disk drive, and LOADS it if found.

The LOAD command can be used within a BASIC program to find and RUN the next program on a tape. This is called chaining.

The RELOCATE FLAG determines where in memory a program is loaded. A relocate flag of 0 tells the Commodore 16 to load the program at the start of the BASIC program area, and a flag of 1 tells it to LOAD from the point where it was SAVEd. The default value of the relocate flag is 0. This is generally used only when loading machine language programs.

NEW

NEW

This command erases the entire program in memory and clears out any variables that may have been used. Unless the program was stored somewhere, it is lost until you type it in again. Be careful when you use this command.

The NEW command can also be used as a statement in a BASIC program. When your Commodore 16 gets to this line, the program is erased and everything stops. This is not especially useful under normal circumstances.

RENAME

RENAME [Ddrive #,] "old name" TO "new name" [,Uunit #]

Used to rename a file on a diskette.

EXAMPLE:

RENAME D0,"ASSET" TO "LIABILITY"	Changes the name of the file from ASSET to LIABILITY

RENUMBER

RENUMBER [new starting line # [,increment [,old starting line #]]]

The new starting line is the number of the first line in the program after renumbering. It defaults to 10.

The increment is the spacing between line numbers, i.e. 10, 20, 30 etc. It also defaults to 10.

The old starting line number is the line number in the program where renumbering is to begin. This allows you to renumber a portion of your program. It defaults to the first line of your program.

This command can only be executed from direct mode.

EXAMPLES:

RENUMBER 20, 20, 1	Starting at line 1, renumbers the program. Line 1 becomes line 20, and other lines are numbered in increments of 20.
RENUMBER, , 65	Starting at line 65, renumbers in increments of 10. Line 65 becomes line 10 (unless there are already lines numbered 10-64, in which case the command is not carried out).

RUN

RUN [line #]

Once a program has been typed into memory or LOADed, the RUN command makes it start working. RUN clears all variables in the program before starting program execution. If there is no number following the command RUN, the computer starts with the lowest numbered program line. If there is a number following the RUN command execution starts at that line. RUN may be used within a program.

EXAMPLES:

RUN	Starts program working from lowest line number.
RUN 100	Starts program at line 100.

SAVE

SAVE ("filename" [,device# [,EOT flag]]]

This command stores a program currently in memory onto a cassette tape or disk. If you just type the word SAVE and press **RETURN**, your Commodore 16 attempts to store the program on the cassette tape. It has no way of checking if there is already a program on the tape in that location, so be careful with your tapes. If you type the SAVE command followed by a name in quotes or a string variable name, the Commodore 16 gives the program that name, so it may be more easily located and retrieved in the future. If you want to specify a device number for the SAVE, follow the name by a comma (after the quotes) and a number or numeric variable. Device number 1 is the tape drive, and number 8 is the disk. After the number on a tape command, there can be a comma and a second number, which is between 0 and 3. If second number is 2, the Commodore 16 puts an END-OF-TAPE marker (EOT flag) after your program. If you are trying to LOAD a program and the Commodore 16 finds one of these markers rather than the program you are trying to LOAD, you get a FILE NOT FOUND ERROR.

EXAMPLES:

SAVE	Stores program to tape without a name.
SAVE "MONEY"	Stores on tape with the name MONEY
SAVE A$	Stores on tape with name in variable A$
SAVE "YOURSELF",8	Stores on disk with name YOURSELF

SAVE "GAME", 1, 2	Stores on tape with name GAME and places an END-OF-TAPE marker after the program.

SCRATCH

SCRATCH "file name" [,**D** drive #] [,**U** unit #]

Deletes a file from the disk directory. As a precaution, you are asked "Are you sure?" before your Commodore 16 completes the operation. Type a Y to perform the SCRATCH or type N to cancel the operation. Use this command to erase unwanted files, to create more space on the disk.

EXAMPLE:

SCRATCH "MY BACK", D1	Erases the file MY BACK from the disk in drive 1

VERIFY

VERIFY "filename" [,device#] [,relocate flag]

This command causes your Commodore 16 to check the program on tape or disk against the one in memory. This is proof that the program you just SAVEd is really saved, to make sure that nothing went wrong. This command is also very useful to position a tape so that your computer resumes writing following the end of the last program on the tape. All you do is tell the Commodore 16 to VERIFY the name of the last program on the tape. It will do so, and tell you that the programs don't match (which you already knew). Now the tape is where you want it, and you can store the next program without fear of erasing an old one.

VERIFY without anything after the command causes the Commodore 16 to check the next program on tape, regardless of its name, against the program now in memory. VERIFY followed by a program name (in quotes) or a string variable searches the tape for that program and then checks. VERIFY followed by a name and a comma and a number checks the program on the device with that number (1 for tape, 8 for disk). The relocate flag is the same as in the LOAD command.

EXAMPLE:

VERIFY	Checks the next program on the tape.
VERIFY "REALITY"	Searches for REALITY on tape, checks against memory.
VERIFY "ME",8,1	Searches for ME on disk, then checks.

BOX

BOX [colour source #], a1, b1, [a2, b2] [,angle] [,paint]

colour source	Colour source (0-3); default is 1 (foreground colour)
a1, b1	Corner coordinate (scaled)
a2, b2	Corner opposite a1, b1 (scaled); default is the PC
angle	Rotation in clockwise degrees; default is 0 degrees
paint	Paint shape with colour (0:off, 1:on); default is 0

This command allows you to draw a rectangle of any size anywhere on the screen. To get the default value, include a comma without entering a value. Rotation is based on the centre of the rectangle. The Pixel Cursor (PC) is left at a2, b2 after the BOX statement is executed.

EXAMPLES:

BOX 1, 10, 10, 60, 60	Draws the outline of a rectangle

BOX , 10, 10, 60, 60, 45, 1	Draws a filled, rotated box (a diamond)
BOX , 30, 90, , 45, 1	Draws a filled, rotated polygon

CHAR

CHAR [colour source #],x,y, "string" [,reverse flag]

colour source ..... Colour source (0 - 3)
x ..... Character column (0 - 39)
y ..... Character row (0 - 24)
"string" ..... Text to be printed
reverse ..... Reverse field flag (0 = off, 1 = on)

Text (alphanumeric strings) can be displayed on any screen at a given location by the CHAR command. Character data is read from the Commodore 16 character ROM area. You supply the x and y coordinates of the starting position and the text string you want to display. Colour source and reverse imaging are optional.

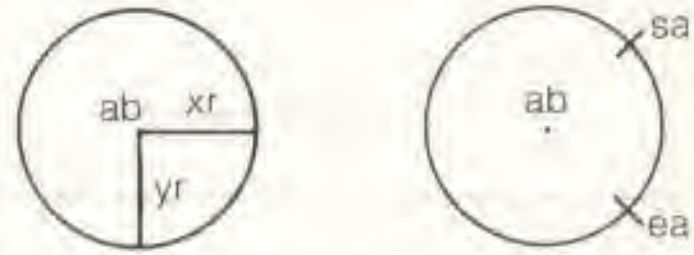
The string is continued on the next line if it attempts to print past the right edge of the screen. When used in TEXT mode, the string printed by the CHAR command works just like a PRINT string, including reverse field, cursors, flash on/off, etc. These control functions inside the string do not work when the CHAR command is used to display text in GRAPHIC mode.

CIRCLE

CIRCLE [cs] ,[a,b], xr [, [yr] [, [sa] [, [ea] [, [angle] [,inc]]]]]

cs ..... Colour source (0 - 3)
a,b ..... Centre coordinate (scaled)
(defaults to the Pixel Cursor [PC])
xr ..... X radius (scaled)
yr ..... Y radius (default is xr)

sa ..... Starting arc angle (default 0)
ea ..... Ending arc angle (default 360)
angle ..... Rotation in clockwise degrees
(default is 0 degrees)
inc ..... Degrees between segments
(default is 2 degrees)



With the CIRCLE command you can draw a circle, ellipse, arc, triangle or an octagon. The final coordinate is on the circumference of the circle at the ending arc angle. Any rotation is about the centre. Arcs are drawn from the starting angle clockwise to the ending angle. The segment increment controls the coarseness of the shape, with lower values for inc creating rounder shapes.

EXAMPLES:

CIRCLE , 160,100,65,10	Draws an ellipse.
CIRCLE , 160,100,65,50	Draws an oval.
CIRCLE , 60,40,20,18,,,,45	Draws an octagon.
CIRCLE , 260,40,20,,,,,90	Draws a diamond.
CIRCLE , 60,140,20,18,,,,120	Draws a triangle.

CLOSE

CLOSE file #

This command completes and closes any files used by OPEN statements. The number following the word CLOSE is the file number to be closed.

EXAMPLE:

CLOSE 2	Logical file 2 is closed.

CLR

CLR

This command erases any variables in memory, but leaves the program itself intact. This command is automatically executed when a RUN or NEW command is given, or when any editing is performed.

CMD

CMD file # [,write list]

CMD sends the output which normally would go to the screen (i.e. PRINT statement, LISTS, but not POKEs into the screen) to another device instead. This could be a printer, or a data file on tape or disk. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file.

EXAMPLES:

OPEN 1,4	OPENS device #4, which is the printer.
CMD 1	All normal output now goes to the printer.
LIST	The LISTing goes to the printer, not the screen - even the word READY.

PRINT#1	Set output back to the screen.
CLOSE 1	Close the file.

COLOR

COLOR source #, colour # [,luminance #]

Assigns a colour to one of the 5 colour sources:

Number	Source
0	background
1	foreground
2	multicolour 1
3	multicolour 2
4	border

Colours you can use are in the range 1 - 16 (1 is black, 2 is white, 9 is orange, etc. from your keyboard colour keys). As an option, you can include the luminance level 0 - 7, with 0 being lowest and 7 being highest. Luminance defaults to 7. Luminance lets you select from eight levels of brightness for any colour except black.

DATA

DATA list of constants separated by commas

This statement is followed by a list of items to be used by READ statements. The items may be numbers or words, and are separated by commas. Words need not be inside of quote marks, unless they contain any of the following characters: SPACE, colon, or comma. If two commas have nothing between them, the value will be READ as a zero for a number, or an empty string. The DATA statement must be part of a program, otherwise it will not be recognized. Also see the RESTORE statment, which allows your Commodore 16 to reread data.

EXAMPLE:

DATA 100, 200, FRED, "WILMA", , 3, 14, ABC123

DEF FN

DEF FN name (variable) = expression

This command allows you to define a complex calculation as a function. In the case of a long formula that is used several times within a program, this can save a lot of space.

The name you give the numeric function begins with the letters FN, followed by any legal numeric variable name. First you must define the function by using the statement DEF followed by the name you've given the function. Following the name is a set of parentheses () with a numeric variable (in this case, X) enclosed. Then you have an equal sign, followed by the formula you want to define. You can "call" the formula, substituting any number for X, using the format shown in line 20 of the example below:

EXAMPLE:

```
10 DEF FNA(X)=12*(34.75-X/.3)+X
20 PRINT FNA(7)
```

The number 7 is inserted each place X is located in the formula given in the DEF statement.

NOTE: DEF FN can only be used with standard numeric functions, not integer or string functions.

DIM

DIM variable (subscripts) [,variable(subscripts)] . . .

Before you can use an array of variables, the program must first execute a DIM statement to establish the DIMensions of that array (unless there are 11 or fewer elements in the array). The statement DIM is followed by the name of the array, which may be any legal variable name. Then, enclosed in parentheses, you put the number (or numeric variable) of elements in each dimension. An array with more than one dimension is called a matrix. You may use any number of dimensions, but keep in mind that the whole list of variables you are creating takes up space in memory, and it is easy to run out of memory if you get carried away. To figure the number of variables created with each DIM, multiply the total number of elements in each dimension of the array. (Each array starts with element 0.)

NOTE: Integer (single-digit) arrays take up 2/5ths of the space of floating point arrays.

EXAMPLE:

```
10 DIM A$(40),B7(15),CC%(4,4,4)
```

41 Elements 16 Elements 125 Elements

You can dimension more than one array in a DIM statement by separating the arrays by commas. If the program executes a DIM statement for any array more than once, you'll get a re'DIMed array error message. It is good programming practice to place DIM statements near the beginning of the program.

DO (LOOP) WHILE (UNTIL EXIT)

DO[**UNTIL** boolean argument | **WHILE** boolean arguement]
statements [**EXIT**]

LOOP[**UNTIL** boolean argument | **WHILE** boolean argument]

(An example of a boolean argument is A=1 or H>=57.)

Performs the statements between the DO statement and the LOOP statement. If no UNTIL or WHILE modifies either the DO or the LOOP statement, execution of the intervening statements continues indefinitely. If an EXIT statement is encountered in the body of a DO loop, execution is transferred to the first statement following the LOOP statement. DO loops may be nested, following the rules defined for FOR-NEXT loops.

If the UNTIL parameter is used, the program continues looping until the boolean argument is satisfied (becomes TRUE). The WHILE parameter is basically the opposite of the UNTIL parameter: the program continues looping as long as the boolean argument is TRUE.

EXAMPLE:

```
DO UNTIL X=0 OR X=1
        :
    LOOP
DO WHILE A$="":GET A$:LOOP
```

DRAW

DRAW [colour source #] [, a1, b1] [**TO** a2, b2,] [. . .]

With this command you can draw individual dots, lines, and shapes. You supply colour source (0-3), starting (a1, b1) and ending points (a2, b2).

EXAMPLES:

a dot: **DRAW** 1, 100, 50 — no endpoint specified, defaults to a1,b1 value for a2,b2 to create a dot

lines: **DRAW** , 10,10, **TO** 100,60
DRAW TO 25,30

a shape: **DRAW** , 10,10 **TO** 10,60 **TO** 100,60 **TO** 10,10

END

END

When the program executes an END statement, the program stops RUNning immediately. You may use the CONT command to re-start the program at the statement following the END statement.

FOR . . . TO . . . STEP

FOR variable = start value **TO** end value [**STEP** increment]

This statement works with the NEXT statement to set up a section of the program that repeats for a set number of times. You may just want your computer to count up to a large number so the program pauses for a few seconds, in case you need something counted, or something must be done a certain number of times (such as printing).

The loop variable is the variable that is added to or subtracted from during the FOR/NEXT loop. The start value and the end value are the beginning and ending counts for the loop variable.

The logic of the FOR statement is as follows. First, the loop variable is set to the start value. When the program reaches a line with the command NEXT, it adds the STEP increment (default = 1) to the value of the loop variable and checks to see if it is higher than the end of loop value. If it is not higher, the next line executed is the statement immediately following the FOR statement. If the loop variable is larger than the end of loop number, then the next statement executed is the one following the NEXT statement. A STEP value can be positive or negative. See also the NEXT statement.

EXAMPLE:

```
10 FOR L = 1 TO 20
20 PRINT L
30 NEXT L
40 PRINT "BLACKJACK! L = "L
```

This program prints the numbers from one to twenty on the screen, followed by the message BLACKJACK! L = 21.

The end of loop value may be followed by the word STEP and another number or variable. In this case, the value following the STEP is added each time instead of one. This allows you to count backwards, by fractions, or any way necessary.

You can set up loops inside one another. This is known as nesting loops. You must be careful to nest loops so that the last loop to start is the first one to end.

EXAMPLE OF NESTED LOOPS:

```
10 FOR L = 1 TO 100
  20 FOR A = 5 TO 11 STEP 2
  30 NEXT A
40 NEXT L
```

This FOR . . . NEXT loop is "nested" inside the larger one.

GET

GET variable list

The GET statement is a way to get data from the keyboard one character at a time. When the GET is executed, the character that was typed is received. If no character was typed, then a null (empty) character is returned, and the program continues without waiting for a key. There is no need to press the **RETURN** key, and in fact the **RETURN** key can be received with a GET.

The word GET is followed by a variable name, usually a string variable. If a numeric were used and any key other than a number was hit, the program would stop with an error message. The GET statement may also be put into a loop, checking for an empty result, which waits for a key to be struck to continue. The GETKEY statement could also be used in this case. This command can only be executed within a program.

EXAMPLE:

```
10 GET A$:IF A$ <> "A" THEN 10
```

This line waits for the 'A' key to be pressed to continue.

GETKEY

GETKEY variable list

The GETKEY statement is very similar to the GET statement. Unlike the GET statement, GETKEY waits for the user to type a character on the keyboard. This lets it be used easily to wait for a single character to be typed.

This command can only be executed within a program.

EXAMPLE:

```
10 GETKEY A$
```

This line waits for a key to be struck. Typing any key will continue the program.

GET#

GET# file number,variable list

Used with a previously OPENed device or file to input one character at a time. Otherwise, it works like the GET statement.

This command can only be executed within a program.

EXAMPLE:

```
GET#1,A$
```

GOSUB

GOSUB line #

This statement is like the GOTO statement, except that your Commodore 16 remembers where it came from. When a line with a **RETURN** statement is encountered, the program jumps back to the statement immediately following the GOSUB. The target of a GOSUB statement is called a subroutine. A subroutine is useful if there is a routine in your program that can be used by several different portions of the program. Instead of duplicating the section of program over and over, you can set it up as a subroutine, and GOSUB to it from the different parts of the program. See also the **RETURN** statement.

EXAMPLE:

```
20 GOSUB 800
```

means go to the subroutine beginning at line 800 and execute it.

```
800 PRINT "HI THERE":RETURN
```

GOTO or GO TO

GOTO line #

After a GOTO statement is executed, the next line to be executed will be the one with the line number following the word GOTO. When used

In direct mode, GOTO line # allows you to start execution of the program at the given line number without clearing the variables.

EXAMPLE:

```
10 PRINT "REPETITION IS THE MOTHER OF LEARNING"
20 GOTO 10
```

The GOTO in line 20 causes line 10 to be run continuously, until the **RUN/STOP** key is pressed.

GRAPHIC

GRAPHIC mode [,clear option]

This statement puts your Commodore 16 in one of its 5 graphic modes:

mode	description
0	normal text
1	high-resolution graphics
2	high-resolution graphics, split screen
3	multicolour graphics
4	multicolour graphics, split screen

When executed, GRAPHIC 1 - 4 allocates a 10K bit-mapped area, and the BASIC text area is moved down below the hi-res area. This area remains allocated even if the user returns to TEXT mode (GRAPHIC 0). If 1 is given in the GRAPHIC statement as the second argument, the screen is also cleared.

EXAMPLES:

`GRAPHIC 1,1`	Selects hi-res graphic mode and clears the screen.
`GRAPHIC 4,0`	Selects multi-colour graphics with an area for text, without clearing the screen.

GRAPHIC CLR

GRAPHIC CLR

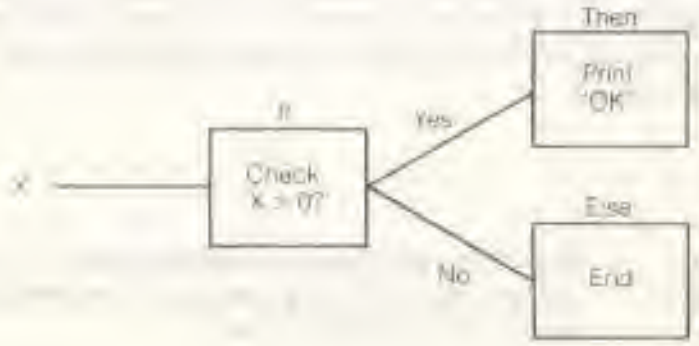
This is actually a form of the GRAPHIC statement; instead of specifying a mode to use graphics, this statement clears the 10k of memory allocated to the graphic area, and that memory space becomes available for BASIC once again.

IF . . . THEN . . :ELSE

IF expression **THEN** then-clause [:**ELSE** else-clause]

IF . . . THEN lets the computer analyze a BASIC expression preceded by IF and take one of two possible courses of action. If the expression is true, the statement following THEN is executed. This expression may be any BASIC statement. If the expression is false, the program goes directly to the next line, unless an ELSE clause is present. The expression being evaluated may be a variable or formula, in which case it is considered true if nonzero, and false if zero. In most cases, there is an expression involving relational operators (=, <, >, <=, >=, <>, AND, OR, NOT).

The ELSE clause, if present, must be in the same line as the IF-THEN part. When an ELSE clause is present, it is executed when the THEN clause isn't executed. In other words, the ELSE clause executes when the IF expression is FALSE.



EXAMPLE:

```
50 IF X>0 THEN PRINT"OK":
   ELSE END
```

Checks the value of X. If X is greater than 0, the THEN clause is executed, and the ELSE clause isn't. If X is not greater than 0, the ELSE clause is executed and the THEN clause isn't.

INPUT

INPUT ["prompt string";] variable list

The INPUT statement allows the computer to ask for data from the person running the program and place it into a variable or variables. The program stops, prints a question mark (?) on the screen, and waits for the person to type the answer and press the **RETURN** key.

The word INPUT is followed by a variable name or list of variable names separated by commmas. There may be a message inside quotes before the list of variables to be input. If this message (called a prompt) is present, there must be a semicolon (;) after the closing quote of the prompt. When more than one variable is to be INPUT, they should be separated by commas when typed in. If not, the computer asks for the remaining values by printing two question marks (??). If you press the **RETURN** key without INPUTting values, the INPUT variables retain the values previously held for those variables. This statement can only be executed within a program.

EXAMPLE:

```
10 INPUT "WHAT'S YOUR NAME";A$
20 INPUT "AND YOUR FAVORITE COLOR";B$
30 INPUT "WHAT'S THE AIR SPEED OF A SWALLOW";A
```

INPUT

INPUT# file number, variable list

This works like INPUT, but takes the data from a previously OPENed file or device. No prompt string is allowed. This command can only be used in program mode.

EXAMPLE:

```
INPUT#2, A$, C, D$
```

LET

[**LET**] variable = expression

The word LET is hardly ever used in programs, since it is not necessary, but the statement itself is the heart of all BASIC programs. Whenever a variable is defined or given a value, LET is always implied. The variable name which is to get the result of a calculation is on the left side of the equal sign, and the number or formula is on the right side.

EXAMPLE:

```
10 LET A = 5

20 B = 6

30 C = A * B + 3

40 D$ = "HELLO"
```

LET is implied (but not necessary) in lines 20, 30 and 40.

LOCATE

LOCATE x-coordinate, y-coordinate

The LOCATE command lets you put the pixel cursor (PC) anywhere on the screen. The PC is the current location of the starting point of the next drawing. Unlike the regular cursor, you can't see the PC, but you can move it with the LOCATE command. For example:

```
LOCATE 160, 100
```

positions the PC in the centre of the high resolution screen. You won't see anything until you actually draw something. You can find out where the PC is at any time by using the RDOT(0) function to get the X-coordinate and RDOT(1) to get the Y-coordinate. The colour source of the dot at the PC can be found by PRINTing RDOT(2). (In all drawing commands where a colour option is available, you may select a value from 0 to 3, corresponding to the background, foreground, multicolour 1, or multicolour 2 as the colour source.)

MONITOR

MONITOR

This command takes you out of BASIC into the built-in machine language monitor program. The monitor is used to develop, debug, and execute machine language programs more easily than from BASIC. See the section on monitor commands for more information. (When in the monitor, typing an 'X' and pressing **RETURN** gets you back to BASIC.)

NEXT

NEXT [variable, . . .,variable]

The NEXT statement is used with the FOR statement. When the computer encounters a NEXT statement, it goes back to the corresponding FOR statement and checks the loop variable. (See FOR statement for more detail.) If the loop is finished, execution proceeds with the statement after the NEXT statement. The word NEXT may be followed by a variable name, a list of variable names separated by commas, or no variable names. If there are no names listed, the last loop started is the one being completed. If the variables are given, they are completed in order from left to right.

EXAMPLE:

```
10 FOR L = 1 TO 10:NEXT
20 FOR L = 1 TO 10:NEXT L
30 FOR L = 1 TO 10:FOR M = 1 TO 10: NEXT M, L
```

ON

ON expression <**GOTO/GOSUB**> line #1 [, line #2, . . .]

This command can make the GOTO and GOSUB statements into special versions of the IF statement. The word ON is followed by a formula, then either GOTO or GOSUB, and a list of line numbers separated by commas. If the result of the calculation of the formula (expression) is 1, the first line in the list is executed. If the result is 2, the second line number is executed, and so on. If the result is 0, or larger than the number of line numbers in the list, the next line executed is the statement following the ON statements. If the number is negative, an ILLEGAL QUANTITY ERROR results.

EXAMPLE:

```
10 INPUT X:IF X<0 THEN 10
20 ON X GOTO 50, 30, 30, 70
25 PRINT "FELL THROUGH":GOTO 10
30 PRINT "TOO HIGH":GOTO 10
50 PRINT"TOO LOW":GOTO 10
70 END
```

When X = 1, ON sends control to the first line number in the list (50). When X = 2, ON sends control to the second line (30), etc.

OPEN

OPEN file # [,device # [,secondary address [,"filename, type, mode"]]]

The OPEN statement allows your Commodore 16 to access devices such as the Datassette recorder and disk for data, a printer, or even the monitor screen. The word OPEN is followed by a logical file number, which is the number to which all other BASIC statements will refer. This number is from 1 to 255. There is normally a second number after the first called the device number. Device number 0 is the Commodore 16 keyboard, 3 is the screen, 1 is the Datassette recorder (default), 4 is the printer, 8 is usually the disk. A zero (0) may be included in front of the device number digit (e.g., 08 for 8, which are interchangeable as far as your Commodore 16 is concerned). It is often a good idea to use the same file number as the device number because it makes it easy to remember which is which. Following the second number may be a third number called the secondary address. In the case of the cassette, this can be 0 for read, 1 for write, and 2 for write with end-of-tape marker at the end. In the case of the disk, the number refers to the channel number. In the printer, the secondary addresses are used to set the mode of the printer. See the Commodore 16 Programmer's Reference Manual or the manual for each specific device for more information on secondary addresses. There may also be a string following the third number, which could be a command to the disk drive or the name of the file on tape or disk. The type and mode refer

to disk files only. (File types are prg,seq, rel, and usr; modes are read and write.)

EXAMPLES:

10 OPEN 3,3	OPENs the SCREEN as a device.
10 OPEN 1,0	OPENs the keyboard as a device.
20 OPEN 1,1,0,"UP"	OPENs the cassette for reading, file to be searched for is named UP.
OPEN 4,4	OPENs a channel to use the printer.
OPEN 15,8,15	OPENs the command channel on the disk.
5 OPEN 8,8,12,"TESTFILE,SEQ,WRITE"	creates a sequential disk file for writing.

See also: **CLOSE, CMD, GET#, INPUT#**, and **PRINT#** statements, system variable **ST**, **DS**, and **DS$**.

PAINT

PAINT [colour source] [,[a,b] [,mode]]

Colour source . . . (0-3); (default is 1, foreground colour)
a,b . . . . . starting coordinate, scaled (default is at the PC)

mode. . . . . .0 = paint an area defined by the colour source selected
1 = paint an area defined by any non-background source

The PAINT command lets you fill an area with colour. It fills in the area around the specified point until a boundary of the same colour (or any non-background colour, depending on which mode you have chosen) is encountered. The final position of the PC will be at the starting point (a,b).

NOTE: If the starting point is already the colour of colour source you name (or any non-background when mode 1 is used), there is no change.

EXAMPLE:

10 CIRCLE , 160,100,65,50	draws outline of circle
20 PAINT , 160,100	fills in the circle with colour

POKE

POKE address, value

The POKE command allows you to change any value in the Commodore 16 RAM memory, and lets you modify many of the Commodore 16 Input/Output registers. POKE is always followed by two numbers, (or equations). The first number is a location inside your computer's memory. This could have any value from 0 to 65535. The second number is a value from 0 to 255, which is placed in the location, replacing any value that was there previously. This command can be used to control anything on the screen, from placing a character at that location to changing the colour there.

EXAMPLE:

10 POKE 16000,8	Sets location 16000 to 8.

20 POKE 16*1000,27 Sets location 16000 to 27.

Note: PEEK, a function related to POKE, is listed under FUNCTIONS.

PRINT

PRINT printlist

The PRINT statement is the major output statement in BASIC. While the PRINT statement is the first BASIC statement most people learn to use, there are many subtleties to be mastered here as well. The word PRINT can be followed by any combinations of these items, which is considered the printlist:

Characters inside of quotes	("text lines")
Variable names	(A, B, A$, X$)
Functions	(SIN(23), ABS(33))
Punctuation marks	(; ,)

The characters inside of quotes are often called literals because they are printed exactly as they appear. Variable names have the value they contain (either a number or a string) printed. Functions also have their number values printed. Punctuation marks are used to help format the data neatly on the screen. The comma divides the screen into 4 columns for data, while the semicolon doesn't add any spaces. Either mark can be used as the last symbol in the statement. This results in the next PRINT statement acting as if it is continuing the last PRINT statement.

EXAMPLE:

	RESULT
10 PRINT "HELLO"	HELLO
20 A$="THERE":PRINT "HELLO,"A$	HELLO,THERE
30 A=4:B=2:PRINT A+B	6
50 J=41:PRINT J;:PRINT J-1	41 40
60 C=A+B:D=A-B:PRINT A;B;C,D	4 2 6 2

See also: POS( ), SPC( ), and TAB( ) FUNCTIONS.

PRINT

PRINT# file #, print list

There are a few differences between this statement and the PRINT. First of all, the word PRINT# is followed by a number, which refers to the device or data file previously OPENed. The number is followed by a comma, and a list of things to be PRINTed. The semicolon acts in the same manner for spacing as it does in the PRINT statement. The comma will send 10 spaces to most printers and can be used as a separator for disk files (see the Programmer's Reference Guide and the Disk Drive Manual). Some devices may not work with TAB and SPC.

EXAMPLE:

100 **PRINT#1,"HELLO THERE!",A$,B$,**

PRINT USING

PRINT [#filenumber,] **USING** format list; print list

These statements let you define the format of string and numeric items you want to print to the screen, printer, or another device. Put the format you want in quotes. This is the format list. Then add a semicolon and a list of what you want printed in the format for the print list. The list can be variables or the actual values you want printed. For example:

```
5 X=32: Y=100.23: A$="CAT"
10 PRINT USING "$##.##";13.25,X,Y
20 PRINT USING "###>#";"CBM",A$
```

When you RUN this, line 10 prints out:

$13.25 $32.00 $*****

prints ***** instead of Y value because Y has 5 digits, which does not conform to format list (as explained below)

Line 20 prints this:

CBM CAT leaves three spaces before printing "CBM" as defined in format list

CHARACTER	NUMERIC	STRING
Hash Sign (#)	X	X
Plus (+)	X	
Minus (-)	X	
Decimal Point (.)	X	
Comma (,)	X	
Dollar Sign ($)	X	
Four Carets (↑↑↑↑)	X	
Equal Sign (=)		X
Greater Than Sign (>)		X

The hash sign (#) reserves room for a single character in the output field. If the data item contains more characters than you have # in your format field, the following occurs:

For a numeric item, the entire field is filled with asterisks (*). No numbers are printed.

For example:

```
10 PRINT USING "####";X
```

For these values for X, this format displays:

A = 12.34 12

A = 567.89 568

A = 123456 ****

For a STRING item, the string data is truncated at the bounds of the field. Only as many characters are printed as there are hash signs (#) in the format item. Truncation occurs on the right.

The plus (+) and minus (-) signs can be used in either the first or last position of a format field but not both. The plus sign is printed if the number is positive. The minus sign is printed if the number is negative.

If you use a minus sign and the number is positive, a blank is printed in the character position indicated by the minus sign.

If you don't use either a plus or minus sign in your format field for a numeric data item, a minus sign is printed before the first digit or dollar symbol if the number is negative and no sign is printed if the number is positive. This means that you can print one character more if the number is positive. If there are too many digits to fit into the field specified by the # and + / - signs, then an overflow occurs and the field is filled with asterisks (*).

A decimal point (.) symbol designates the position of the decimal point in the number. You can only have one decimal point in any format field. If you don't specify a decimal point in your format field, the value is rounded to the nearest integer and printed without any decimal places.

When you specify a decimal point, the number of digits preceding the decimal point (including the minus sign, if the value is negative) must not exceed the number of # before the decimal point. If there are too many digits an overflow occurs and the field is filled with asterisks (*).

A comma (,) lets you place commas in numeric fields. The position of the comma in the format list indicates where the comma appears in a printed number. Only commas within a number are printed. Unused commas to the left of the first digit appear as the filler character. At least one # must precede the first comma in a field.

If you specify commas in a field and the number is negative, then a minus sign is printed as the first character even if the character position is specified as a comma.

EXAMPLES:

FIELD	EXPRESSION	RESULT	COMMENT
##.#+	-.01	0.01-	Leading zero added.
##.#-	1	1.0	Trailing zero added.
####	-100.5	-101	Rounded to no decimal places.
####	-1000	****	Overflow because four digits and minus sign cannot fit in field.

###.	10	10.	Decimal point added.
#$##	1	$1	Leading $ sign.

A dollar sign ($) symbol shows that a dollar sign will be printed in the number. If you want the dollar sign to float (always be placed before the number), you must specify at least one # before the dollar sign. If you specify a dollar sign without a leading #, the dollar sign is printed in the position shown in the format field.

If you specify commas and/or a plus or minus sign in a format field with a dollar sign, your program prints a comma or sign before the dollar sign.

The four up arrows or carets (↑↑↑↑) symbol is used to specify that the number is to be printed in E + format. You must use # in addition to the ↑↑↑↑ to specify the field width. The ↑↑↑↑ must appear after the # in the format field.

You must specify four carets (↑↑↑↑) when you want to print a number in E -format (scientific notation). If you specify more than one but fewer than four carets, you get a syntax error. If you specify more than four carets only the first four are used. The fifth caret (and subsequent carets) are interpreted literally as no text symbols.

An equal sign (=) is used to centre a string in the field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is centreed in the field. If the string contains more characters than can be fit into the field. The right-most characters are truncated and the string fills the entire field.

A greater than sign (>) is used to right justify a string in a field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is right justified in the field. If the string contains more characters than can be fit into the field, the right-most characters are truncated and the string fills the entire field.

PUDEF

PUDEF "1 through 4 characters"

PUDEF lets you redefine up to 4 symbols in the **PRINT USING** statement. You can change blanks, commas, decimals points, and dollar signs into some other character by placing the new character in the correct position in the **PUDEF** control string.

Position 1 is the filler character. The default is a blank. Place a new character here when you want another character to appear in place of blanks.

Position 2 is the comma character. Default is a comma.

Position 3 is the decimal point.

Position 4 is the dollar sign.

EXAMPLES:

```
10 PUDEF " *"          space   PRINTs * in the place of blanks.
20 PUDEF "/&"                  PRINTs & in place of commas.
30 PUDEF ",."          space   PRINTs decimal points in place of
                               commas, and commas in place of
                               decimal points.
40 PUDEF "!,£"                 PRINTs English pound sign in place of $.
```

READ

READ variable list

This statement is used to get information from DATA statements into variables, where the data can be used. The READ statement variable list may contain both strings and numbers. Care must be taken to avoid reading strings where the READ statement expects a number, which produces an ERROR message.

EXAMPLE:

```
READ A$, G$, Y
```

REM

REM message

The REMark is just a note to whoever is reading a LIST of the program. It may explain a section of the program, give information about the author, etc. REM statements in no way effect the operation of the program, except to add to its length (and therefore slow it down). The word REM may be followed by any text, although use of graphic characters gives strange results.

EXAMPLE:

10 NEXT X: REM THIS LINE IS UNNECESSARY

RESTORE

RESTORE [line #]

When executed in a program, the pointer to the item in a DATA statement which is to be read next is reset to the first item in the list. This gives you the ability to re-READ the information. If a [line #] follows the RESTORE statement, the pointer is set to that line. Otherwise the pointer is reset to the first DATA statement in the program.

EXAMPLE:

RESTORE 200

RESUME

RESUME [line # | **NEXT**]

Used to return to execution after TRAPping an error. With no arguments, RESUME attempts to re-execute the line in which the error occurred. RESUME NEXT resumes execution at the next statement following the statement containing the error; RESUME line # will GOTO the specific line and begin execution there.

RETURN

RETURN

This statement is always used with the GOSUB statement. When the program encounters a RETURN statement, it goes to the statement immediately following the last GOSUB command executed. If no GOSUB was previously issued, then a RETURN WITHOUT GOSUB ERROR message is delivered, and program execution is stopped.

SCALE

SCALE <1/0>

The scaling of the bit maps in multicolour and high resolution modes can be changed with the SCALE command. Entering:

SCALE 1

turns scaling on. Coordinates may then be scaled from 0 to 1023 in both X and Y rather than the normal scale values, which are:

multicolour mode	X = 0 to 159	Y = 0 to 199
high resolution mode	0 to 319	0 to 199

Scaling can be turned off by entering 'SCALE 0'.

SCNCLR

SCNCLR

Clears the current screen, whether graphics, text, or both (split screen).

SOUND

SOUND voice #, frequency control, duration

This statement produces a SOUND using one of three voices with a frequency control in the range 0 - 1023 for a duration of 0 - 65535 60ths of a second.

V	Voice
1	Voice 1 (tone)
2	Voice 2 (tone)
3	Voice 2 (white noise)

If a SOUND for voice N is requested, and the previous SOUND for the same N is still playing, BASIC waits for the previous SOUND to complete. SOUND with a duration of 0 is a special case. It causes BASIC to turn off the current SOUND for that voice immediately, regardless of the time remaining on the previous SOUND. See the MUSIC NOTE TABLE in the appendix for the frequency control values that correspond to real notes.

EXAMPLE:

SOUND 2, 800, 3600

Plays a note using voice 2 with frequency set at 800 for one minute

SSHAPE/GSHAPE

SSHAPE and GSHAPE are used to save and restore rectangular areas of multicolour or high resolution screens using BASIC string variables. The command to save an area is:

SSHAPE string variable, a1,b1 [,a2,b2]

string variable . . .String name to save data in
a1,b1 . . . . . .Corner coordinate (scaled)
a2,b2 . . . . . .Corner coordinate opposite (a1,b1) (default is the PC)

Because BASIC limits string lengths to 255 characters, the size of the area you may save is limited. The string size required can be calculated using one of the following (unscaled) formulas:

L(mcm) = INT ((ABS(a1-a2) + 1) / 4 + .99) * (ABS(b1-b2) + 1) + 4

L(h-r) = INT ((ABS(a1-a2) + 1) / 8 + .99) * (ABS(b1-b2) + 1) + 4

(mcm) refers to multi-colour mode; (h-r) is high resolution.

The shape is saved row by row. The last four bytes of the string contain the column and row lengths less one (i.e.: ABS (a1-a2)) in low/high byte format (if scaled divide the lengths by 3.2 (X) and 5.12 (Y)).

The command to display a saved shape on any area of the screen:

GSHAPE string variable name [, [a,b] [,mode]]

string . . . . . .Contains shape to be drawn
a,b . . . . . .Top left coordinate telling where to draw the shape (scaled - the default is the PC)
mode . . . . . .Replacement mode:
0: place shape as is (default)
1: place field inverted shape
2: OR shape with area
3: AND shape with area
4: XOR shape with area

EXAMPLES:

SSHAPE "VARIABLE$", 0, 0	Saves screen area from the upper left corner to where the cursor is positioned under the name VARIABLE$.
GSHAPE "VARIABLE$",,,1	Displays VARIABLE$ shape with background and foreground colours reversed, with the top left of the shape positioned at the cursor.

STOP

STOP

This statement halts the program. A message, BREAK IN LINE #, where the # is the line number containing the STOP. The program can be re-started at the statement following STOP if you use the CONT command. The STOP statement is usually used while debugging a program.

SYS

SYS address

The word SYS is followed by a decimal number or numeric variable in the range 0 to 65535. The program begins executing the machine language program starting at that memory location. This is similar to the USR function, but does not pass a parameter. See the Programmer's Reference Guide for information about machine language programs.

TRAP

TRAP [line#]

When turned on, TRAP intercepts all error conditions (including the RUN/STOP key) except "UNDEF'D STATEMENT ERROR". In the event of any execution error, the error flag is set, and execution is transferred to the line number named in the TRAP statement. The line number in which the error occurred can be found by using the system variable EL. The specific error condition is contained in system variable ER. The string function ERR$(ER) gives the error message corresponding to any error condition ER.

NOTE: An error in a TRAP routine cannot be trapped. The RESUME statement can be used to resume execution. TRAP with no line# argument turns off error TRAPping.

TRON

TRON

TRON is used in program debugging. This statement begins trace mode. When you are in trace mode, as each statement executes, the line number of that statement is printed.

TROFF

TROFF

This statement turns trace mode off.

VOL

VOL volume level

Sets the current VOLume level for SOUND commands. VOLume may be set from 0 to 8, where 8 is maximum volume, and 0 is off. VOL affects both voices.

WAIT

WAIT address, value 1 [, value 2]

The WAIT statement is used to halt the program until the contents of a location in memory changes in a specific way. The address must be in the range from 0 to 65535. Value 1 and value 2 must be in the range from 0 to 255.

The content of the memory location is first exclusive-ORed with value 2 (if present), and then logically ANDed with value 1. If the result is zero, the program checks the memory location again. When the result is not zero, the program continues with the next statement.

Additional Graphic Statement Information

There are a few concepts that apply to all of the bit map graphics statements. First is the concept of the Pixel Cursor (PC). The PC is similar to the cursor in text mode; it is the position where the next dot is to be drawn. Unlike the text cursor, the PC is invisible. All drawing commands use the PC. In addition, the locate command allows you to reposition the PC without drawing anything.

Wherever you would use X,Y coordinates in a drawing command, you can use RELATIVE coordinates instead. Relative coordinates are based on the current value of the PC. To use relative coordinates, just place a + or - in front of your coordinates. A plus sign before the X value moves the PC to the right. A minus sign before the X value moves the PC to the left. Similarly, a minus sign before the Y coordinate moves the PC up, while a plus sign moves the PC down. For example:

LOCATE +100,-25	moves the PC right 100 pixels and up 25.
DRAW1,+10,+10**TO**100,100	draws a line 10 pixels right and 10 pixels below the current value of the PC to the absolute point 100,100.

You can also specify a distance and and angle relative to the current PC by separating the two parameters by a semicolon.

For example:

LOCATE 50;45	moves the PC from its current location by a distance of 50 dots at an angle of 45 degrees.

FUNCTIONS

Numeric Functions

Numeric functions are classified as such because they return numbers. The functions they perform range from calculating mathematical functions to specifying a screen location. Numeric functions follow the form:

FUNCTION (argument)

where the argument can be a numerical value, variable, or string.

ABS(X) (absolute value)

The absolute value function returns the magnitude of the argument X.

ASC(X$)

This function returns the ASCII code (number) of the first character of X$.

ATN(X) (arctangent)

Returns the angle whose tangent is X, measured in radians.

COS(X) (cosine)

Returns the value of the cosine of X, where X is an angle measured in radians.

DEC (hexadecimal-string)

Returns decimal value of hexadecimal-string (0 < hexadecimal-string < FFFF)

EXAMPLE:

```
N=DEC("F4")
```

EXP(X)

Returns the value of the mathematical constant e (2.71828183) raised to the power of X.

FNxx(x)

Returns the value of the user-defined function xx created in a DEF FNxx statement.

INSTR (string 1, string 2 [,starting-position])

Returns position of string 2 in string 1 at or after the [starting-position]. The starting-position defaults to the beginning of string 2. If no match is found, a value of 0 is returned.

EXAMPLE:

```
PRINT INSTR("THE CAT IN THE HAT", "CAT")
```

the result is 5, because CAT starts at the fifth character in string 1

INT(X) (integer)

Returns the integer portion of X, with all decimal places to the right of the decimal point removed. The result is always less-than or equal to X. Thus, any negative numbers with decimal places become the integer less-than their current value (e.g. INT(-4.5)=-5).

If the INT function is to be used for rounding off, the form is INT(X + .5) or INT(X - .5).

EXAMPLE:

```
X=INT(X*100 + .5)/100          Rounds to the next highest penny.
```

JOY (n)

When n = 1 ..... Position of joystick #1
n = 2 ..... Position of joystick #2

Any value of 128 or more means the fire button is also depressed. The direction is indicated as follows:

```
                     UP
fire = 128 +          1
               8            2
    LEFT   7          0          3 RIGHT
               6            4
                      5
                    DOWN
```

EXAMPLE:

JOY(2) with value of 135 fires joystick #2 to the left

LOG(X) (logarithm)

This returns the natural log of X. The natural log is log to the base e (see EXP(X)). To convert to log base 10, divide by LOG(10).

PEEK(X)

This function gives the contents of memory location X, where X is located in the range of 0 to 65535, returning a result from 0 to 255. This is often used in conjunction with the POKE statement.

RCLR(N)

Returns current colour assigned to source N (0 =< N =< 4) (0=background, 1=foreground, 2=multicolour 1, 3=multicolour 2, 4=border).

RDOT(N)

Returns information about the current position of the pixel cursor (PC) at XPOS/YPOS.

N - 0 for XPOS
    1 for YPOS
    2 colour source

RGR(X)

Returns current graphic mode (X is a dummy argument and can be any value.)

RLUM(N)

Returns current luminance level assigned to colour source N

RND(X) (random number)

This function returns a random number between 0 and 1. This is useful in games, to simulate dice rolls and other elements of chance, and is also used in some statistical applications. The first random number should be generated by the formula RND(-TI), to start things off differently every time. After this, the number in X should be a 1, or any positive number. (X represents the seed, or what the RaNDom number is based on.) If X is zero, RND is re-seeded from the hardware clock every time RND is used. A negative value for X seeds the random number generator using X and gives a random number sequence. The use of the same negative number for X as a seed results in the same sequence of random numbers. A positive value gives random numbers based on the previous seed.

To simulate the rolling of a die, use the formula INT(RND(1)*6+1). First the random number from 0-1 is multiplied by 6, which expands the range to 0-6 (actually, greater than zero and less then six). Then 1 is added, making the range 1 to under 7. The INT function chops off all the decimal places, leaving the result as a digit from 1 to 6.

To simulate 2 dice, add two of the numbers obtained by the above formula together.

EXAMPLE:

```
100 X=INT(RND(1)*6)+INT(RND(1)*6)+2    Simulates 2 dice.
100 X=INT(RND(1)*1000)+1               Number from 1-1000.
100 X+INT(RND(1)*150)+100              Number from 100-249.
```

SGN(X) (sign)

This function returns the sign, as in positive, negative, or zero, of X. The result is + 1 if positive, 0 if zero, and - 1 if negative.

SIN(X) (sine)

This is the trigonometric sine function. The result is the sine of X, where X is an angle in radians.

SQR(X) (square root)

This function returns the square root of X, where X is a positive number or 0. If X is negative, an ILLEGAL QUANTITY ERROR results.

TAN(X) (tangent)

This gives the tangent of X, where X is an angle in radians.

USR(X)

When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations 1281 and 1282. The parameter X is passed to the machine language program in the floating point accumulator. Another number is passed back to the BASIC program through the calling variable. In other words, this allows you to exchange a variable between machine code and BASIC. See the Programmer's Reference Guide for more details on this, and on machine language programming.

VAL(X$)

This function converts the string X$ into a number, and is essentially the inverse operation from STR$. The string is examined from the left-most character to the right, for as many characters as are in recognizable number format. If the Commodore 16 finds illegal characters, only the portion of the string up to that point is converted.

EXAMPLE:

```
10 X=VAL("123.456")          X=123.456
10 X=VAL("3E03")             X=3000
10 X=VAL("12A13B")           X=12
10 X=VAL("RIUO17*")          X=0
10 X=VAL("-1.23.23.23")      X=-1.23
```

String Functions

String functions differ from numeric functions in that they return characters, graphics or numbers from a string (defined by quotation marks) instead of a number.

CHR$(X)

This function returns a string character whose ASCII code is X.

ERR$(N)

Returns string describing error condition N (see TRAP)

HEX$(N)

Returns a 4 character string containing the hexadecimal representation of value N ($0 < N < 65535$)

LEFT$(X$,X)

This returns a string containing the leftmost X characters of X$.

LEN(X$)

Returns the number of characters (including spaces and other symbols) in the string X$.

MID$(X$,N,X)

This returns a string containing X characters, starting from the Nth character in X$. MID$ can also be used on the left side of assignment statement as a pseudo-variable as well as a function. MID$ (string variable, starting position, length) = source string.

This function reassigns values of positions (starting position) through (starting position + length) of source string to the characters of string variable in corresponding locations. Length defaults to the length of string variables, and an error results if (starting position + length) is greater than the length of the source string.

EXAMPLE:

```
10 A$="THE LAST GOODBYE":
20 PRINT A$
30 MID$(A$,6,3)="ONG"
40 PRINT A$
```

RIGHT$(X$,X)

This returns the right-most X characters in X$.

STR$(X)

This returns a string which is identical to the PRINTed version of X$.

EXAMPLE:

```
A$=STR$(X)
```

Other Functions

FRE (X)

This function returns the number of unused bytes available in memory. X is a dummy argument.

POS (X)

This function returns the number of the column (0–39) where the next PRINT statement begins on the screen. X is a dummy argument.

SPC (X)

This is used in the PRINT statement to skip over X spaces. X can have a value from 0-255.

TAB (X)

This is used in the PRINT statement. The next item to be printed is in column number X. X can have a value from 0 to 255.

π (PI)

The pi symbol, when used in an equation, has the value of 3.14159265.

VARIABLES & OPERATORS

Variables

Your Commodore 16 uses three types of variables in BASIC. These are: normal numeric, integer numeric, and string (alphanumeric) variables.

Normal **NUMERIC VARIABLES**, also called **floating point variables**, can have any value from 10^{-38} to 10^{+38}, with up to nine digits of accuracy. When a number becomes larger than nine digits can show, as in 10^{-10} or 10^{+10}, your computer displays it in scientific notation form, with the number normalized to 1 digit and eight decimal places, followed by the letter E and the power of ten by which the number is multiplied. For example, the number 12345678901 is displayed as 1.234356789E+10.

INTEGER VARIABLES can be used when the number is from +32767 to -32768, and with no fractional portion. An integer variable is a number like 5, 10, or -100. Integers take up less space than floating point variables when used in an array.

STRING VARIABLES are those used for character data, which may contain numbers, letters, and any other character that your Commodore 16 can make. An example of a string variable is "COMMODORE 16".

VARIABLE NAMES

Variable names may consist of a single letter, a letter followed by a number, or two letters. Variable names may be longer than 2 characters, but only the first two are significant.

An integer variable is specified by using the percent (%) sign after the variable name. String variables have the dollar sign ($) after their names.

EXAMPLES:

Numeric Variable Names: A, A5, BZ

Integer Variable Names: A%, A5%, BZ%

String Variable Names: A$, A5$, BZ$

ARRAYS are lists of variables with the same name, using an extra number (or numbers) to specify an element of the array. Arrays are defined using the DIM statement, and may be floating point, integer, or string variables arrays. The array variable name is followed by a set of parentheses ( ) enclosing the number of the variable in the list.

EXAMPLES: A(7),BZ%(11),A$(87)

Arrays may have more than one dimension. A two dimensional array may be viewed as having rows and columns, with the first number identifying the column and the second number in the parentheses identifying the row (as if specifying a certain grid on a map).

EXAMPLES: A(7,2),BZ%(2,3,4),Z$(3,2)

RESERVED VARIABLE NAMES
There are seven variable names which are reserved for use by the Commodore 16, and may not be used for another purpose. These are the variables DS, DS$, ER, EL, ST,TI, and TI$. You also can't use KEYWORDS such as TO and IF, or any names that contain KEYWORDS, such as SRUN, RNEW, or XLOAD as variable names.

ST is a status variable for input and output (except normal screen/keyboard operations). The value of ST depends on the results of the last input/output operation. A more detailed explanation of ST is in the Series 264 Programmer's Reference Guide, but in general, if the value of ST is 0 the operation was sucessful

TI and TI$ are variables that relate to the real-time clock built into your Commodore 16. The system clock is updated every 1/60th of a second. It starts at 0 when your Commodore 16 is turned on, and is reset only by changing the value of TI$. The variable TI gives you the current value of the clock in 1/60ths of a second.

TI$ is a string that reads the value of the real-time clock as a 24 hour clock. The first two characters of TI$ contain the hour, the 3rd and 4th characters are the minutes, and the 5th and 6th characters are the seconds. This variable can be set to any value (so long as all characters are numbers), and will be automatically updated as a 24 hour clock.

EXAMPLE: TI$ = "101530" sets the clock to 10:15 and 30 seconds (AM)

The value of the clock is lost when your Commodore 16 is turned off. It starts at zero when your computer is turned on, and is reset to zero when the value of the clock exceeds 235959 (23 hours, 59 minutes and 59 seconds).

The variable DS reads the disk drive command channel, and returns the current status of the drive. To get this information in words, PRINT DS$. These status variables are used after a disk operation, like a DLOAD or DSAVE, to find out why the red error light on the disk drive is blinking.

ER, EL, and ERR$ are variables used in error trapping routines. They are usually only useful within a program. ER returns the last error encountered since the program was RUN. EL is the line where the error occured. ERR$ is a function which allows your program to print one of the BASIC error messages. PRINT ERR$(ER) prints out the proper error message.

BASIC OPERATORS

The ARITHMETIC operators include the following signs:

+ addition
- subtraction
* multiplication
/ division
↑ raising to a power (exponentiation)

On a line containing more than one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows: First, exponentiation, then multiplication and division, and last, addition and subtraction. If two operations have the same priority, then calculations are performed in order from left to right. If you want these operations to occur in a different order, Commodore 16 BASIC allows you to give a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. You have to make sure that your equations have the same number of left parentheses as right parentheses, or you will get a SYNTAX ERROR message when your program is run.

There are also operators for equalities and inequalities, called RELATIONAL operators. Arithmetic operators always take priority over relational operators.

=	is equal to
<	is less than
>	is greater than
<= or =<	is less than or equal to

>= or => is greater than or equal to
<> or >< is not equal to

Finally, there are three LOGICAL operators, with lower priority than both arithmetic and relational operators:

AND
OR
NOT

These are used most often to join multiple formulas in IF . . . THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and -)

EXAMPLES:

IF A=B AND C=D THEN 100	requires both A=B & C=D to be true
IF A=B OR C=D THEN 100	allows either A=B or C=D to be true
A=5:B=4:PRINT A=B	displays a value of 0
A=5:B=4:PRINT A>B	displays a value of -1
PRINT 123 AND 15:PRINT 5 OR 7	displays 11 and 7

BASIC Abbreviation and Reference Chart

KEYWORD	ABBREVIATION			TYPE
ABS	a	SHIFT	B	function—numeric
ASC	a	SHIFT	S	function—numeric
ATN	a	SHIFT	T	function—numeric
AUTO	a	SHIFT	U	command
BACKUP	b	SHIFT	A	command
BOX	b	SHIFT	O	statement
CHAR	ch	SHIFT	A	statement
CHR$	c	SHIFT	H	function—string
CIRCLE	c	SHIFT	I	statement
CLOSE	cl	SHIFT	O	statement
CLR	c	SHIFT	L	statement
CMD	c	SHIFT	M	statement
COLLECT	col	SHIFT	L	command
COLOR	co	SHIFT	L	statement
CONT	c	SHIFT	O	command
COPY	co	SHIFT	P	command
COS		none		function—numeric
DATA	d	SHIFT	A	statement
DEC		none		function—numeric
DEF FN	d	SHIFT	E	statement
DELETE	de	SHIFT	L	command
DIM	d	SHIFT	I	statement
DIRECTORY	di	SHIFT	R	command
DLOAD	d	SHIFT	L	command
DO		none		statement
DRAW	d	SHIFT	R	statement
DSAVE	d	SHIFT	S	command
END	e	SHIFT	N	statement
ERR$	e	SHIFT	R	function—string
EXP	e	SHIFT	X	function—numeric
FOR	f	SHIFT	O	statement
FRE	f	SHIFT	R	function—numeric
GET	g	SHIFT	E	statement
GETKEY	getk	SHIFT	E	statement
GET#		none		statement
GOSUB	go	SHIFT	S	statement
GOTO	g	SHIFT	O	statement
GRAPHIC	g	SHIFT	R	statement

KEYWORD	ABBREVIATION			TYPE
GSHAPE	g	SHIFT	S	statement
HEADER	he	SHIFT	A	command
HEX$	h	SHIFT	E	function—string
IF..GOTO		none		statement
IF..THEN..ELSE		none		statement
INPUT		none		statement
INPUT#	i	SHIFT	N	statement
INSTR	in	SHIFT	S	function—numeric
INT		none		function—numeric
JOY	j	SHIFT	O	function—numeric
KEY	k	SHIFT	E	command
LEFT$	le	SHIFT	F	function—string
LEN		none		function—numeric
LET	l	SHIFT	E	statement
LIST	l	SHIFT	I	command
LOAD	l	SHIFT	O	command
LOCATE	lo	SHIFT	C	statement
LOG		none		function—numeric
LOOP	lo	SHIFT	O	statement
MID$	m	SHIFT	I	function—string
MONITOR	m	SHIFT	O	statement
NEW		none		command
NEXT	n	SHIFT	E	statement
ON..GOSUB	on..go	SHIFT	S	statement
ON..GOTO	on..g	SHIFT	O	statement
OPEN	o	SHIFT	P	statement
PAINT	p	SHIFT	A	statement
PEEK	p	SHIFT	E	function—numeric
POKE	p	SHIFT	O	statement
POS		none		function—numeric
PRINT	?			statement
PRINT#	p	SHIFT	R	statement
PRINT USING	?us	SHIFT	I	statement
PUDEF	p	SHIFT	U	statement
RCLR	r	SHIFT	C	function—numeric
RDOT	r	SHIFT	D	function—numeric
READ	r	SHIFT	E	statement
REM		none		statement

KEYWORD	ABBREVIATION			TYPE
RENAME	re	SHIFT	N	command
RENUMBER	ren	SHIFT	U	command
RESTORE	re	SHIFT	S	statement
RESUME	res	SHIFT	U	statement
RETURN	re	SHIFT	T	statement
RGR	r	SHIFT	G	function—numeric
RIGHT$	r	SHIFT	I	function—string
RLUM	r	SHIFT	L	function—numeric
RND	r	SHIFT	N	function—numeric
RUN	r	SHIFT	U	command
SAVE	s	SHIFT	A	command
SCALE	sc	SHIFT	A	statement
SCNCLR	s	SHIFT	C	statement
SCRATCH	sc	SHIFT	R	command
SGN	s	SHIFT	G	function—numeric
SIN	s	SHIFT	I	function—numeric
SOUND	s	SHIFT	O	statement
SPC(	s	SHIFT	P	function—special
SQR	s	SHIFT	Q	function—numeric
SSHAPE	s	SHIFT	S	statement
STatus		none		reserved—numeric variable
STOP	s	SHIFT	T	statement
STR$	st	SHIFT	R	function—string
SYS	s	SHIFT	Y	statement
TAB(	t	SHIFT	A	function—special
TAN		none		function—numeric
TI		none		reserved—numeric variable
TI$		none		reserved—string variable
TRAP	t	SHIFT	R	statement
TROFF	tro	SHIFT	F	statement
TRON	tr	SHIFT	O	statement
UNTIL	u	SHIFT	N	statement
USR	u	SHIFT	S	function—special
VAL		none		function—numeric
VERIFY	v	SHIFT	E	command
VOL	v	SHIFT	O	statement
WAIT	w	SHIFT	A	statement
WHILE	w	SHIFT	H	statement

APPENDICES

- Error messages
- Disk error messages
- Deriving mathematical functions
- Musical note table
- Screen display and ASCII codes
- Book list

APPENDIX A
Error Messages

These error messages are printed by BASIC. You can also PRINT the messages through the use of the ERR$() function. The error number refers only to the number assigned to the error for use with this function.

ERROR #	ERROR NAME	
1	**TOO MANY FILES**	There is a limit of 10 files OPEN at one time.
2	**FILE OPEN**	An attempt was made to open a file using the number of an already open file.
3	**FILE NOT OPEN**	The file number specified in an I/O statement must be opened before use.
4	**FILE NOT FOUND**	No file with that name exists (disk).
5	**DEVICE NOT PRESENT**	The required I/O device not available.
6	**NOT INPUT FILE**	An attempt made to GET or INPUT data from a file that was specified as output only.
7	**NOT OUTPUT FILE**	An attempt made to send data to a file that was specified as input only.
8	**MISSING FILE NAME**	An OPEN, LOAD, or SAVE to the disk drive generally requires a file name.
9	**ILLEGAL DEVICE NUMBER**	An attempt made to use a device improperly (SAVE to the screen, etc.)
10	**NEXT WITHOUT FOR**	Either loops are nested incorrectly, or there is a variable name in a NEXT statement that doesn't correspond with one in a FOR.
11	**SYNTAX**	A statement is unrecognizable by BASIC. This could be because of missing or extra parenthesis, misspelled keyword, etc.
12	**RETURN WITHOUT GOSUB**	A RETURN statement encountered when no GOSUB statement was active.
13	**OUT OF DATA**	A READ statement encountered, without data left unREAD.
14	**ILLEGAL QUANTITY**	A number used as the argument of a function or statement is outside the allowable range.
15	**OVERFLOW**	The result of a computation is larger than the largest number allowed (1.701411833E+38)
16	**OUT OF MEMORY**	Either there is no more room for program and program variables, or there are too many DO, FOR, or GOSUB statements in effect.
17	**UNDEF'D STATEMENT**	A line number referenced doesn't exist in the program.
18	**BAD SUBSCRIPT**	The program tried to reference an element of an array out of the range specified by the DIM statement.
19	**REDIM'D ARRAY**	An array can only be DIMensioned once. If an array is referenced before that array is DIM'd, an automatic DIM (to 10) is performed.
20	**DIVISION BY ZERO**	Division by zero is not allowed.
21	**ILLEGAL DIRECT**	INPUT or GET statements are only allowed within a program.

22	**TYPE MISMATCH**	This occurs when a number is used in place of a string or vice-versa.
23	**STRING TOO LONG**	A string can contain up to 255 characters.
24	**FILE DATA**	Bad data read from a tape file.
25	**FORMULA TOO COMPLEX**	Simplify the expression (break into 2 parts or use fewer parentheses)
26	**CAN'T CONTINUE**	The CONT command does not work if the program was not RUN, there was an error, or a line has been edited.
27	**UNDEF'D FUNCTION**	A user defined function referenced that was never defined.
28	**VERIFY**	The program on tape or disk does not match the program in memory.
29	**LOAD**	There was a problem loading. Try again.
30	**BREAK**	The stop key was hit to halt program execution.
31	**CAN'T RESUME**	A RESUME statement encountered without TRAP statement in effect.
32	**LOOP NOT FOUND**	The program has encountered a DO statement and cannot find the corresponding LOOP.
33	**LOOP WITHOUT DO**	LOOP encountered without a DO statement active.
34	**DIRECT MODE ONLY**	This command is allowed only in direct mode, not from a program.
35	**NO GRAPHICS AREA**	A command (DRAW, BOX, etc.) to create graphics encountered before the GRAPHIC command was executed.
36	**BAD DISK**	An attempt failed to HEADER a diskette, because the quick header method (no ID) was attempted on an unformatted diskette, or the diskette is bad.

APPENDIX B
Disk Error Messages

These error messages are returned through the DS and DS$ reserved variables.

NOTE: Error message numbers less than 20 should be ignored with the exception of 01, which gives information about the number of files scratched with the SCRATCH command.

20	**READ ERROR** (block header not found)	The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.
21	**READ ERROR** (no sync character)	The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/writer head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.
22	**READ ERROR** (data block not present)	The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.
23	**READ ERROR** (checksum error in data block)	This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.
24	**READ ERROR** (byte decoding error)	The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.
25	**WRITE ERROR** (write-verify error)	This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.
26	**WRITE PROTECT ON**	This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. Typically, this is caused by using a diskette with a write protect tab over the notch.
27	**READ ERROR** (checksum error in header)	The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.
28	**WRITE ERROR** (long data block)	The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad diskette format (the data extends into the next block), or by hardware failure.
29	**DISK ID MISMATCH**	This message is generated when the controller has been requested to access a diskette which has not been initialized. The message can also occur if a diskette has a bad header.
30	**SYNTAX ERROR** (general syntax)	The DOS cannot interpret the command sent to the command channel. Typically, this is caused

		by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.
31	**SYNTAX ERROR** (invalid command)	The DOS does not recognize the command. The command must start in the first position.
32	**SYNTAX ERROR** (invalid command)	The command sent is longer than 58 characters.
33	**SYNTAX ERROR** (invalid file name)	Pattern matching is invalidly used in the OPEN or SAVE command.
34	**SYNTAX ERROR** (no file given)	The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon(:) has been left out of the command.
39	**SYNTAX ERROR** (invalid command)	This error may result if the command sent to command channel (secondary address 15) is unrecognized by the DOS.
50	**RECORD NOT PRESENT**	Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.
51	**OVERFLOW IN RECORD**	PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.
52	**FILE TOO LARGE**	Record position within a relative file indicates that disk overflow will result.
60	**WRITE FILE OPEN**	This message is generated when a write file that has not been closed is being opened for reading.
61	**FILE NOT OPEN**	This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.
62	**FILE NOT FOUND**	The requested file does not exist on the indicated drive.
63	**FILE EXISTS**	The file name of the file being created already exists on the diskette.
64	**FILE TYPE MISMATCH**	The file type does not match the file type in the directory entry for the requested file.
65	**NO BLOCK**	This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use.

66	**ILLEGAL TRACK AND SECTOR**	The DOS has attempted to access a track or block which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.
67	**ILLEGAL SYSTEM T OR S**	This special error message indicates an illegal system track or sector.
70	**NO CHANNEL** (available)	The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.
71	**DIRECTORY ERROR**	The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem reinitialize the diskette to restore the BAM in memory. Some active files may be terminated by the corrective action. NOTE: BAM = Block Availability Map
72	**DISK FULL**	Either the blocks on the diskette are used or the directory is at its entry limit. DISK FULL is sent when two blocks are available on the 1541 to allow the current file to be closed.
73	**DOS MISMATCH** (73, CBM DOS V2.6 1541)	DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. (A utility routine is available to assist in converting from one format to another.) This message may also appear after power up.
74	**DRIVE NOT READY**	An attempt has been made to access the Floppy Disk Drive without any diskette present.

APPENDIX C
Deriving Mathematical Functions

Functions that are not intrinsic to BASIC 3.5 may be calculated as follows:

FUNCTION	BASIC EQUIVALENT
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X)=−ATN(X/SQR (−X*X+1))+π/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X−1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X−1)) +(SGN(X)−1*π/2)
INVERSE COTANGENT	ARCCOT(X)=ATN(X)+π/2
HYPERBOLIC SINE	SINH(X)=(EXP(X)−EXP(−X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)−+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(−X)/(EXP(x)+EXP (−X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(−X)/(EXP(X) −EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR (−X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR (x*x+1)/x))
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(x−1))/2

APPENDIX D
Musical Note Table

NOTE	SOUND REGISTER VALUE	ACTUAL FREQUENCY (HZ)
A	7	110
B	118	123.5
C	169	130.8
D	262	146.8
E	345	164.7
F	383	174.5
G	453	195.9
A	516	220.2
B	571	246.9
C	596	261.4
D	643	293.6
E	685	330
F	704	349.6
G	739	392.5
A	770	440.4
B	798	494.9
C	810	522.7
D	834	588.7
E	854	658
F	864	699
G	881	782.2
A	897	880.7
B	911	989.9
C	917	1045
D	929	1177
E	939	1316
F	944	1398
G	953	1575

The above table contains the sound register values of four octaves of notes. The sound register values are used as the second parameter of the SOUND command. To use the first note in the table (A—sound

register value 7) use the 7 as the second number after the SOUND command—SOUND 1,7,30.

Use the following formula to find the sound register values for frequencies other than those in the table:

SOUND REGISTER VALUE = 1024—(111860.781/FREQUENCY)

Both the table of sound register values and the above formula are for NTSC televisions. This is the television standard used throughout the United States and all of Canada. If you are in a country where PAL is the television standard, you should use the following formula to calculate new sound register values for the entire table.

SOUND REGISTER VALUE = 1024—(111840.45/FREQUENCY)

APPENDIX E
Screen Display Codes

The following chart lists all of the characters built into the Commodore character sets. It shows which numbers should be POKEd into screen memory (locations 3072 to 4095) to get a desired character. (Remember to set color memory—2048 to 3071.) Also shown is which character corresponds to a number PEEKed from the screen.

Two character sets are available, but only one set at a time. This means that you cannot have characters from one set on the screen at the same time you have characters from the other set displayed. The sets are switched by holding down the SHIFT and C= keys simultaneously.

From BASIC, PRINT CHR$(142) will switch to upper-case/graphics mode and PRINT CHR$(14) switches to upper/lower-case mode.

Any number on the chart may also be displayed in REVERSE. The reverse character code may be obtained by adding 128 to the values shown.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	T	t	20	(		40
A	a	1	U	u	21	)		41
B	b	2	V	v	22	*		42
C	c	3	W	w	23	+		43
D	d	4	X	x	24	,		44
E	e	5	Y	y	25	-		45
F	f	6	Z	z	26	.		46
G	g	7	[		27	/		47
H	h	8	£		28	0		48
I	i	9	]		29	1		49
J	j	10	↑		30	2		50
K	k	11	←		31	3		51
L	l	12	**SPACE**		32	4		52
M	m	13	!		33	5		53
N	n	14	"		34	6		54
O	o	15	#		35	7		55
P	p	16	$		36	8		56
Q	q	17	%		37	9		57
R	r	18	&		38	:		58
S	s	19	'		39	;		59

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
<		60		T	84			108
=		61		U	85			109
>		62		V	86			110
?		63		W	87			111
		64		X	88			112
	A	65		Y	89			113
	B	66		Z	90			114
	C	67			91			115
	D	68			92			116
	E	69			93			117
	F	70			94			118
	G	71			95			119
	H	72	**SPACE**		96			120
	I	73			97			121
	J	74			98			122
	K	75			99			123
	L	76			100			124
	M	77			101			125
	N	78			102			126
	O	79			103			127
	P	80			104			
	Q	81			105			
	R	82			106			
	S	83			107			

Codes from 128-255 are reversed images of codes 0-127

APPENDIX F
ASCII and CHR$ Codes

This appendix shows you what characters will appear if you PRINT CHR$(X), for all possible values of X. It will also show the values obtained by typing PRINT ASC("X"), where X is any character you can type. This is useful in evaluating the character received in a GET statement, converting upper/lower-case, and printing character based commands (like switch to upper/lower-case) that could not be enclosed in quotes.

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	↑	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES SHIFT C=	8		25	*	42	;	59
ENABLES SHIFT C=	9		26	+	43	<	60
	10	ESCAPE	27	,	44	=	61
	11	RED	28	-	45	>	62
	12	→	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	(	33	2	50	C	67

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68	♠	97	π	126	┼	155
E	69	─	98	◥	127	PUR	156
F	70	─	99		128	←	157
G	71	─	100	♠	129	YEL	158
H	72	─	101	FLASH ON	130	CYN	159
I	73	─	102	FLASH OFF	131	SPACE	160
J	74	─	103		132	▌	161
K	75	─	104	f1	133	▀	162
L	76	◪	105	f3	134	▁	163
M	77	◪	106	f5	135	▁	164
N	78	◪	107	f7	136	▁	165
O	79	▁	108	f2	137	▒	166
P	80	◪	109	f4	138	▁	167
Q	81	◪	110	f6	139	▒	168
R	82	▁	111	HELP	140	◪	169
S	83	▁	112	SHIFT RETURN	141	▁	170
T	84	●	113	SWITCH TO UPPER CASE	142	┼	171
U	85	▁	114		143	▄	172
V	86	♥	115	BLK	144	┼	173
W	87	▁	116	↓	145	┼	174
X	88	◪	117	RVS OFF	146	▁	175
Y	89	⊠	118	CLR HOME	147	┼	176
Z	90	○	119	INST DEL	148	┼	177
[	91	♣	120	◪	149	┼	178
£	92	▁	121	⊠	150	┼	179
]	93	♦	122	○	151	▁	180
↑	94	┼	123	♣	152	▌	181
←	95	▒	124	▁	153	▄	182
─	96	─	125	♦	154	▁	183

APPENDIX G

Books For Commodore Products

The following lists include a sampling of the computer and programming books available. The title of the book is listed first, followed by the author and publisher.

Commodore Books

- VIC 20 Programmer's Reference Guide
- Commodore 64 Programmer's Reference Guide
- Commodore Plus/4 Programmer's Reference Guide
- Mastering Your VIC 20
- Four VIC 20 Computer Books:
  - VIC Revealed, Nick Hampshire
  - VIC Games, Nick Hampshire
  - VIC Graphics, Nick Hampshire
  - Stimulating Simulations for the VIC, C.W. Engel
- Introduction to BASIC, Part 1 and 2, Andrew Colin
- Commodore Software Encyclopedia, Third Edition

BASIC Programming

- Armchair BASIC, An Absolute Beginner's Guide to Programming in BASIC, Fox & Fox, Osborne/McGraw-Hill
- BASIC Handbook, Second Edition, Lien, Compusoft
- Basic Commodore 64 BASIC, Coan, Hayden
- Elementary BASIC, Leogard & Singer, SRA
- How to Build a Program, Emmerichs, Dilithium Press
- Instant Freeze-Dried Computer Programming in BASIC, Brown
- My Computer Likes Me When I Speak in BASIC, Albrecht, Dilithium Press
- Nailing Jelly to a Tree, Willis & Danley, Dilithium Press
- The Programmer's Book of Rules, Ledin & Ledin, Lifetime Learning Publishers
- Technical BASIC, Kassab, Prentice-Hall

Machine Language Programming

- Machine Language for Beginners, Mansfield, COMPUTE! Books
- Programming the 6502, Zaks, Sybex
- 6502 Assembly Language Programming, Leventhal, Osborne/McGraw-Hill
- 6502 Micro Chart, Micro Logic Corp.
- 6502 Software Design, Scanlon, Sams
- The 6502 Software Gourmet Guide & Cookbook, Findlay, Hayden

INDEX